FREE! 3 HOURS OF EXCLUSIVE VIDEO TUTORIALS

228 PAGES OF PRO ADVICE

THE RESPONSIVE WEB DESIGN HANDBOOK

VOLUME II

20 PRACTICAL PROJECTS

BUILD BRILLIANT SITES THAT WORK ON ANY DEVICE

Future

FEATURING Design a prototype for a web app ● Create perfect flexible layouts ● Build a responsive WordPress portfolio ● Design sites with responsive images

THE RESPONSIVE WEB DESIGN HANDBOOK

VOLUME II

It's nearly six years ago that Ethan Marcotte's seminal article 'Responsive Web Design' (*netm.ag/marcotte-278*) came out. The way we design sites for multiple devices has evolved a lot since then. But it can still be daunting, and it's not just about layout any more.

So in this new edition of *The Responsive Web Design Handbook* we look at the latest tools and techniques, new workflows and processes, and different layout options. We also dive into how to build images, forms, tables and charts, emails, and WordPress themes that shine on any device.

Want to make it responsive? We've got you covered. There's also an inspirational Showcase section full of case studies from the likes of Amnesty International and the LA Times, and the brightest minds in the industry give their thoughts on web performance, accessibility and plenty of other issues related to responsive design.

And if you're eager for even more, don't miss issue 278 of **net** magazine (*netm.ag/issue-278*), which includes a 20-page responsive design guide, featuring brand new articles by Ben Callahan, Sarah Drasner and Brad Frost.

Enjoy!

Oliver Lindberg, editor
oliver.lindberg@futurenet.com

FEATURED AUTHORS

BEN CALLAHAN

Ben is a founder and the president of Sparkbox. From page 36 he lists 10 indispensable tools for responsive web design
w: *bencallahan.com*
t: @bencallahan

RACHEL ANDREW

Rachel is a web developer, writer and speaker, and one half of the team behind the Perch CMS. On page 144 she introduces the CSS Grid Layout
w: *rachelandrew.co.uk*
t: @rachelandrew

NOAH STOKES

Noah is a partner at Bold in California. On page 84 he explains why responsive design is sucking the magic out of our sites
w: *noahstokes.com*
t: @motherfuton

CLARISSA PETERSON

UX designer and developer Clarissa is author of *Learning Responsive Web Design*. On page 188 she explains how to master responsive type
w: *clarissapeterson.com*
t: @clarissa

CONTENTS


FREE! 3 HOURS OF EXCLUSIVE VIDEO TUTORIALS
228
THE
RESPONSIVE
WEB DESIGN HANDBOOK
VOLUME II
20
BUILD BRILLIANT SITES THAT
WORK ON ANY DEVICE

Contents

FEATURES

SHOWCASE

VOICES

PROJECTS

THE PRO'S GUIDE TO Responsive WEB DESIGN

Advanced techniques to build better sites for any device

AUTHOR

JUSTIN AVERY

By day, Justin (@justinavery) is a technical consultant, and by night he curates the RWD Weekly newsletter, hosts a responsive podcast and runs r*esponsivedesign.is*

ILLUSTRATION

MARTINA FLOR

Berlin-based Martina creates type, lettering and illustration work for clients all over the world *martinaflor.com*

Responsive web design sounds incredibly simple. Opt for flexible grids for the layout, use flexible media (images, video, `iframes`), and apply media queries to update these measurements to best arrange content on any viewport. In practice we've learnt it is not really that easy. Tiny issues that crop up during every project keep us scratching our heads, and occasionally even carving fingernail trenches on our desks.

Since I began curating the Responsive Design Weekly newsletter (*responsivedesignweekly.com*) I've been fortunate enough to speak with many members of the web community and hear their experiences. I wanted to find out exactly what the community really wanted to learn, so I circulated a survey to readers. Here are the top results:

1. Responsive images
2. Improving performance
3. Responsive typography
4. Media queries in JavaScript
5. Progressive Enhancement
6. Layout

With those topics in mind, I ran a series of podcasts (*responsivedesign.is/feeds/podcast*) asking some of our industry leaders for their thoughts. In their responses, one point was unanimous: focus upon getting the basics right before you start worrying about exciting and advanced techniques. By taking things back to the basics, we are able to build a robust interface for everyone, layering on features when the device or user's context allows.

"So ... what about these advanced techniques?" I hear you ask. I think Stephen Hay summed it up best when he said: "The ultimate RWD technique is to start off by not using any advanced RWD techniques. Start with structured content and build your way up. Only add a breakpoint when the design breaks and the content dictates it and ... that's it."

In this article, I'll begin with the basics and add layers of complexity as the situation allows, to build up to those advanced techniques. Let's get started.

RESPONSIVE IMAGES

Fluid media was a key part of RWD when it was first defined by Ethan Marcotte. `width: 100%;` , `max-width: 100%;` still works today, but the responsive image landscape has become a lot more complicated. With increasing numbers of screen sizes, pixel density and devices to support we crave greater control.

The three main concerns were defined by the Responsive Images Community Group (RICG):

1. The kilobyte size of the image we are sending over the wire
2. The physical size of the image we are sending to high DPI devices
3. The image crop in the form of art direction for the particular size of the viewport

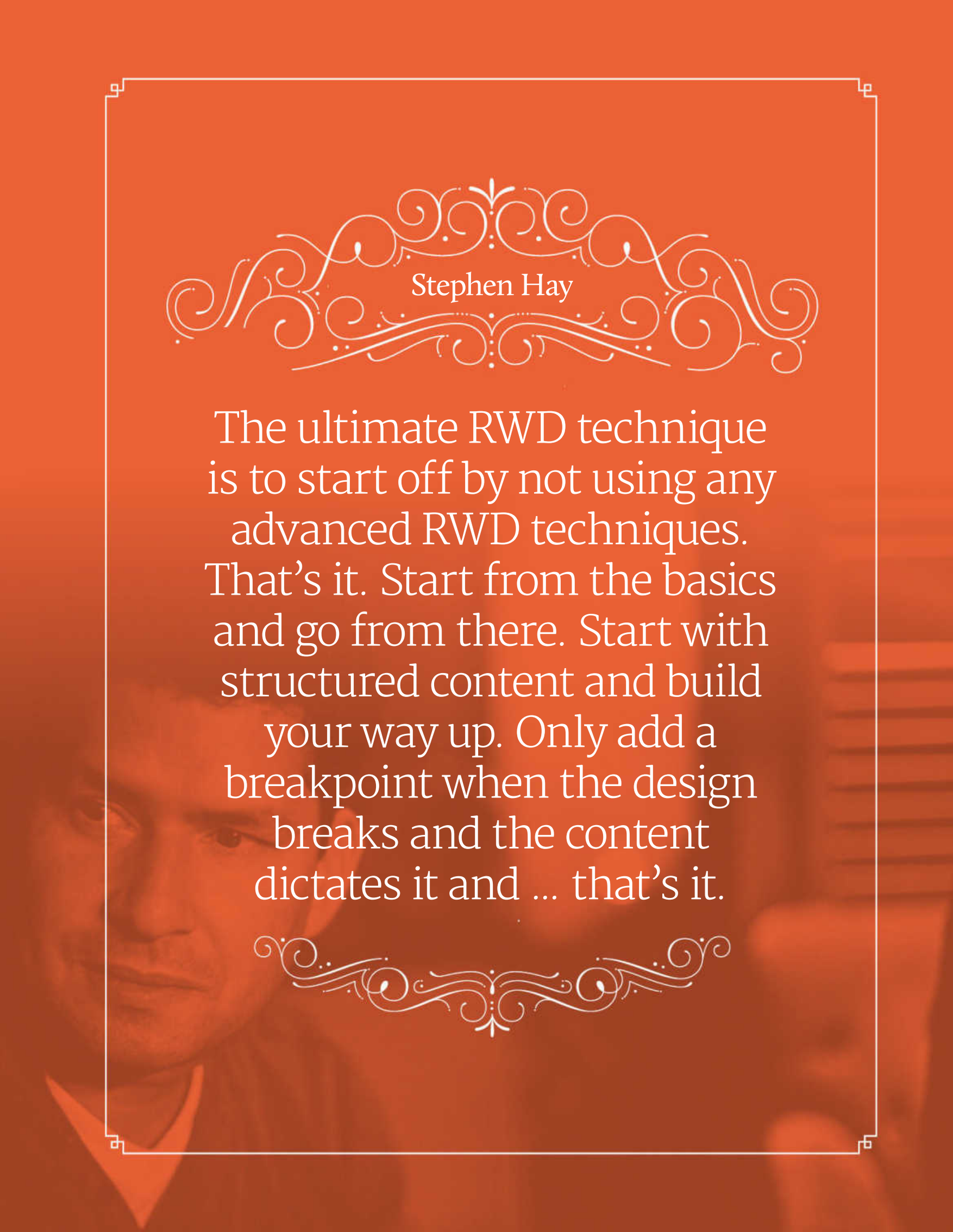
Yoav Weiss, with help from Indiegogo (*netm.ag/blink-255*), has done the majority of the work on the Blink implementation – Google's fork of WebKit, and by the time you're reading this it will be shipped in Chrome and Firefox. Safari 8 will ship `srcset`, however the `sizes` attribute is only in nightly builds and `<picture>` is not yet implemented.

With the arrival of anything new to our web development process, it can be difficult to get started. Let's run through an example step by step.

```
<img
    <!-- Declare the fallback image for all non picture supporting browsers -->
    src="horse-350.jpg"
    <!-- Declare all of the image sizes in srcset. Include the image width using the w descriptor to inform the browser of the width of each image.-->
    srcset="horse-350.jpg 350w,
    horse-500.jpg 500w,
    horse-1024.jpg 1024w,
    horse.jpg 2000w"
    <!-- Sizes inform the browser of our site layout. Here we're saying for any viewport that is 64ems and bigger, use an image that will occupy 70% of the viewport -->
    sizes="(min-width: 64em) 70vw,
    <!-- If the viewport isn't that big, then for any viewport that is 37.5ems and bigger, use an image that occupies 95% of the viewport -->
    (min-width: 37.5em) 95vw,
    <!-- and if the viewport is smaller than that, then use an image that occupies 100% of the viewport-->
    100vw"
    <!-- always have alt text.-->
    alt="A horse" />
```

From a performance point of view it doesn't matter if you use `min-width` or `max-width` in the `sizes` attribute – but the source order does matter. The browser will always use the first matching size.

Also, remember we are hard-coding the `sizes` attribute to be directly defined against our design. ▶



Stephen Hay

The ultimate RWD technique is to start off by not using any advanced RWD techniques. That's it. Start from the basics and go from there. Start with structured content and build your way up. Only add a breakpoint when the design breaks and the content dictates it and … that's it.

Cutting the mustard

'Cutting the mustard' is an approach that involves testing for specific device features. If these features exist then the user is served a more advanced and immersive experience, while those that do not support them will still get the content they were looking for.

The test effectively separates users into two groups; those with HTML5 capabilities and those without. The same initial content is delivered to both sets, however the HTML5 group will also benefit from `localStorage`, `querySelector` and `addEventListener`.

```
if('querySelector' in document
    && 'localStorage' in window
    && 'addEventListener' in window) {
    // bootstrap the javascript application
    }
```

Cutting the mustard allows us to organise users into tiers based on feature detection. These tiers all receive a different website experience, but every user still has access to the basic content.

There's a useful BBC article at *netm.ag/mustardarticle-260*, or check out a presentation by the BBC's Tom Maslen – who first coined the term 'cutting the mustard – at *netm.ag/mustardvid-260*.

This may cause issues moving forwards. For example, if you redesign your site, you'll need to revisit all of the `<img>` or `<picture>`s and redefine the sizes. If you are using a CMS, this can be overcome as part of your build process.

WordPress already has a plugin to help (*netm.ag/picture-260*). It defines the `srcset` on WP standard image varieties and allows you to declare `sizes` in a central location. When the page is generated from the database, it swaps out any mentions on `<img>` and replaces them with the `picture` markup.

Basic

- Think about whether you really need to include an image. Is the image core content, or is it decorative? One less image will mean a faster load time
- Optimise the images you do need to include using ImageOptim (*imageoptim.com*)
- Set expire headers for your images on your server or .htaccess file (see details under 'Performance')
- PictureFill (*github.com/scottjehl/picturefill*) provides polyfill support for pictures

Advanced

- Lazy load your images using jQuery's Lazy Load plugin (*appelsiini.net/projects/lazyload*)
- Use `HTMLImageElement.Sizes` and `HTMLPictureElement` for feature detection.
- The advanced PictureFill WP plugin (*netm.ag/imgtags-260*), will allow you to define custom image widths and sizes values

PERFORMANCE

To get the fastest perceived performance on our pages, we need all the HTML and CSS required to render the top part of our page within the first response from the server. That magic number is 14kb and is based on the max congestion window size within the first round-trip time (RTT). Patrick Hamann, frontend technical lead at the *Guardian* (*theguardian.com*), and his team have managed to break the 1000ms barrier using a mixture of frontend and backend techniques. The *Guardian*'s approach is to ensure the required content – the article – is delivered to the user as quickly as possible and within the 14kb magic number.

Let's look at the process:

1. User clicks on a Google link to a news story
2. A single blocking request is sent to the database for the article. No related stories or comments are requested
3. The HTML is loaded containing Critical CSS in the `<head>`
4. A 'Cut the mustard' process is undertaken and any conditional elements based upon the user's device features are loaded
5. Any content related to or supporting the article itself (related article images, article comments and so on) are lazy loaded into place

Optimising the critical rendering path like this means the `<head>` is 222 lines long. However, the critical content the user came to see still comes within the 14kb initial payload when gzipped. It's this process that helps break that 1000ms rendering barrier.

Conditional and lazy loading

Conditional loading improves the user's experience based on their device feature. Tools like Modernizr allow you to test for these features, but be aware that just because a browser says it offers support, that doesn't always mean full support.

One technique is to hold off loading something until the user shows intent to use that feature. This would be considered conditional. You can hold off loading in the social icons until the user hovers over or touches the icons, or you could avoid loading an `iframe` Google Map on smaller viewports where the user is likely to prefer linking to a dedicated mapping application. Another approach is to 'Cut the mustard' – see boxout above for details on this.

Lazy loading is defined as something that you always intend to load on the page – images that are a part of the article, comments or other related articles – but that don't need to be there for the user to start consuming the content.

Basic

- Enable gzipping for files and set expire headers for all static content (*netm.ag/expire-260*)
- Use the Lazy Load jQuery plugin (*appelsiini.net/projects/lazyload*). This loads images when approaching the viewport, or after a certain delay

Advanced

- Set up Fastly (*fastly.com*) or CloudFlare (*cloudflare.com*). Content delivery networks (CDNs) deliver your static content to users faster than your own server, and have some free tiers
- Enable SPDY for http2-enabled browsers to take advantage of http2 features like parallel http requests
- Social Count (*filamentgroup.com/lab/socialcount*) allows for conditional loading of your social icons
- Using the Static Maps API will allow you to switch out Interactive Google maps for images. Take a look at Brad Frost's example at *netm.ag/static-260*
- Ajax Include Pattern (*netm.ag/ajax-260*) will load content snippets from either a `data-before`, `data-after` or `data-replace` attribute

RESPONSIVE TYPOGRAPHY

Typography is about making your content easy to digest. Responsive typography extends this to ensure readability across a wide variety of devices and viewports. Jordan Moore admits that type is one thing he isn't willing to budge on. Drop an image or two if you need, but make sure you have great type.

Stephen Hay suggests setting the HTML font size to 100 per cent (read: just leave it as it is) because each browser or device manufacturer makes a reasonably readable default for a particular resolution or device. For most desktop browsers this is 16px.

On the other hand, Moore uses the REM unit and HTML `font-size` to set a minimum font size for certain content elements. For example, if you want the byline of an article to always be 14px, then set that as the base `font-size` on the HTML element and set `.byline { font-size: 1rem;}`. As you scale the `body: font-size:` to suit the viewport you will not impact the `.by-line` style.

A good reading line length is also important – aim for 45 to 65 characters. There's a bookmarklet you can use to check your content (*netm.ag/bookmarklet-260*).

If you are supporting multiple languages with your design, then line length may vary as well. Moore suggests using `:lang(de) article {max-width: 30em}` to cover off any issues there.

To maintain vertical rhythm, set `margin-bottom` against content blocks, `<ul>`, `<ol>`, `<blockquote>`, `<table>`, `<blockquote>` and so on, to the same as your `line-height`. If the rhythm is interrupted with the introduction of images you could fix it by adding Baseline.js (*netm.ag/baseline-260*) or BaselineAlign.js (*netm.ag/align-260*).

Basic

- Base your font on 100 per cent body
- Work in relative em units
- Set your margins to your line height to maintain vertical rhythm in your design

Advanced

- Improve font loading performance with Enhance.js (*github.com/filamentgroup/enhance*) or deferred font loading (*netm.ag/defer-260*)
- Use Sass `@includes` for semantic headings. Often we need to use the `h5` style in a sidebar widget that requires `h2` markup. Use Bearded's Typographic Mixins (*netm.ag/mixin-260*) to control the size and remain semantic with the below code:

```
.sidebar h2 {
    @include heading-5
  }
```

MEDIA QUERIES IN JAVASCRIPT

Ever since we have been able to control the layout across a variety of viewports through media queries, we've been looking for a way to tie that into running our JavaScript as well. There are a few ways to fire JavaScript on certain viewport widths, heights and orientations, and some smart people have written some easy-to-use native JS plugins like Enquire.js (*netm.ag/enquire-241*) and Simple State Manager (*netm.ag/simple-260*). You could even build this yourself using `matchMedia`. However, you have the issue that you need to duplicate your media queries in your CSS and JavaScript.

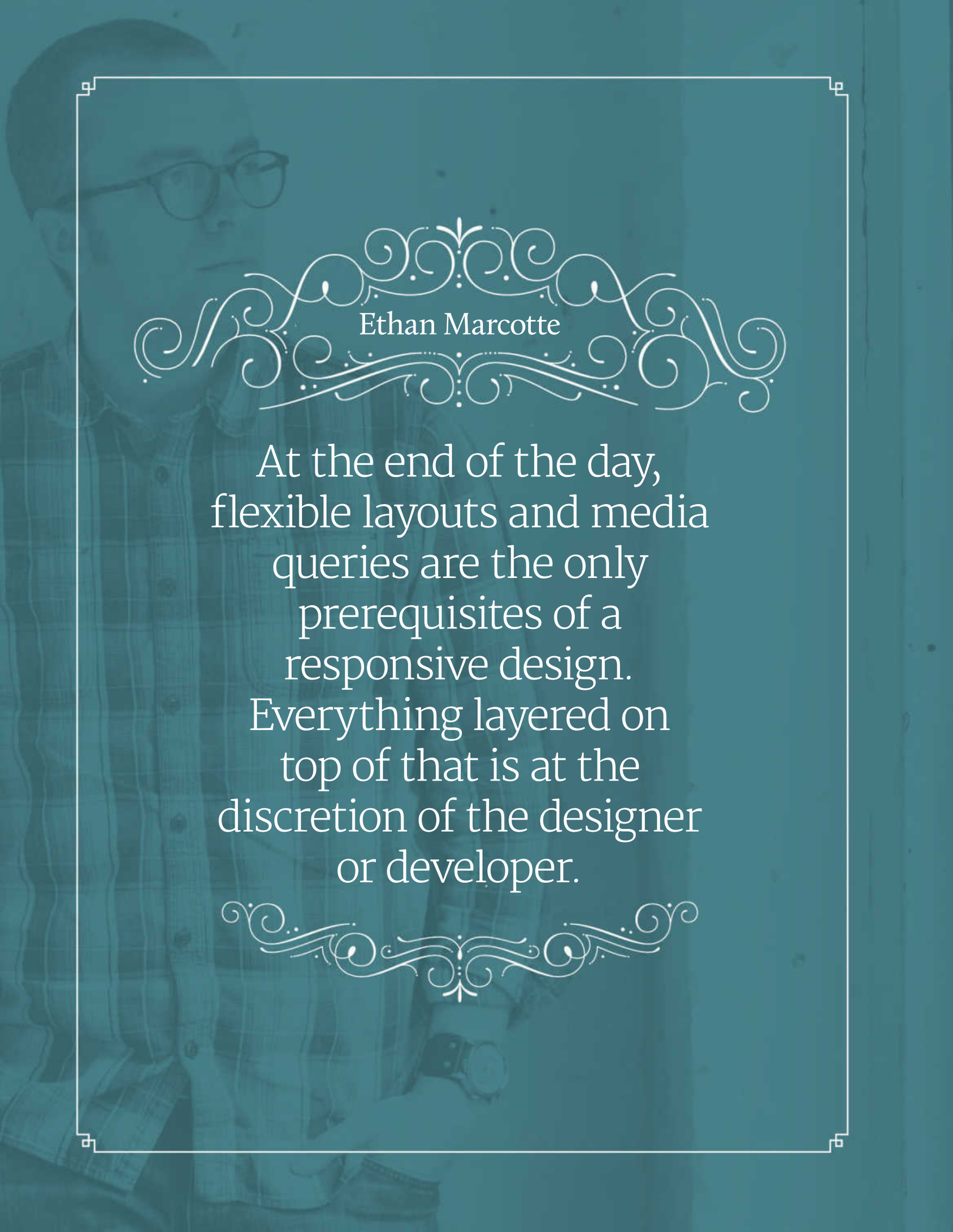
Responsibly responsive

Web designer **Scott Jehl** considers if RWD is bad for performance

I think the web in general has a problem with bloated pages, and that problem both preceded and continues alongside responsive sites. In that light, I don't think it's fair to attribute the performance issues found in many responsive sites to RWD itself, but rather to a broader lack of follow-through with widely-recommended optimisations. RWD is defined as nothing more than a means for adapting visual layout, and it's the best approach we have for doing that.

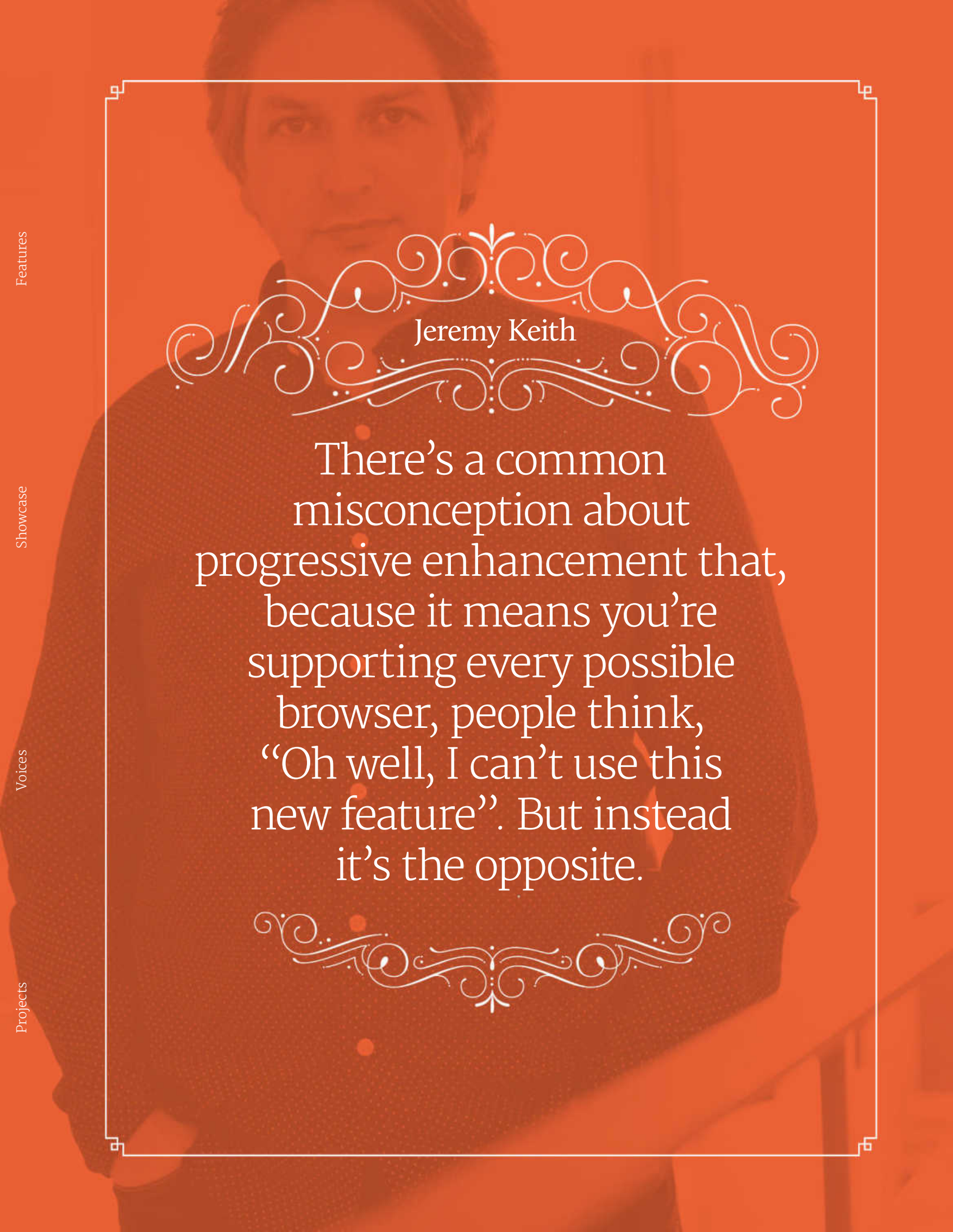
All of the performance recommendations we need to follow for building sites in general – concatenating, minifying or gzipping code, optimising imagery, caching and so on – should be applied to responsive sites as well. Sometimes it's tricky to pull off with multi-device code, but I think it's more of a challenge than an impossibility.

Responsible Responsive Design by Scott Jehl is available at netm.ag/responsible-260



Ethan Marcotte

At the end of the day, flexible layouts and media queries are the only prerequisites of a responsive design. Everything layered on top of that is at the discretion of the designer or developer.



Jeremy Keith

There's a common misconception about progressive enhancement that, because it means you're supporting every possible browser, people think, "Oh well, I can't use this new feature". But instead it's the opposite.

Aaron Gustafson has a neat trick that means you don't have to manage and match your media queries in your CSS and your JS. The idea originally came from Jeremy Keith (*adactio.com/journal/5429*) and in this example Gustafson has taken it to a full implementation.

Using `getActiveMQ` (*netm.ag/media-260*), inject `div#getActiveMQ-watcher` at the end of the body element and hide it. Then within the CSS set `#getActiveMQ-watcher {font-family: break-0;}` to the first media query, `font-family: break-1;` to the second, `font-family: break-2;` to the third and so on.

The script uses `watchResize()` (*netm.ag/resize-260*) to check to see if the size of the viewport has changed, and then reads back the active `font-family`. Now you can use this to hook JS enhancements like adding a tabbed interface to a `<dl>` when the viewport allows, changing the behaviour of a lightbox, or updating the layout of a data table. Check out the `getActiveMQ` Codepen at *netm.ag/active-260*.

Basic

- Forget about JavaScript for different viewports. Provide content and website functions to users in a way they can access it across all viewports. We should never *need* JavaScript

Advanced

- Extend Gustafson's method by using Breakup (*github.com/BPScott/breakup*) as a predefined list of media queries and automating the creation of the list of font families for `getActiveMQ-watcher`

PROGRESSIVE ENHANCEMENT

A common misconception about progressive enhancement is that people think, "Oh well I can't use this new feature", but really, it's the opposite. Progressive enhancement means that you can deliver a feature if it's only supported in one or even no browsers, and over time people will get a better experience as new versions arrive.

If you look at the core function of any website, you can deliver that as HTML and have the server side do all the processing. Payments, forms, Likes, sharing, emails, dashboards – it can all be done. Once the basic task is built we can then layer the awesome technologies on top of that, because we have a safety net to catch those that fall through. Most of the advanced approaches we have talked about here are based upon progressive enhancement.

LAYOUT

Flexible layout is simple to say, but it has many different approaches. Create simple grid layouts with less markup by using `:nth-child()` selector.

```
/* declare the mobile first width for the grid */
.grid-1-4 { float:left; width: 100%; }

/* When the viewport is at least 37.5em then set the grid to 50% per element */
@media (min-width: 37.5em) {
  .grid-1-4 { width: 50%; }

/* Clear the float every second element AFTER the first. This targets the 3rd, 5th, 7th, 9th... in the grid.*/
  .grid-1-4:nth-of-type(2n+1) { clear: left; }
}

@media (min-width: 64em) { .grid-1-4 { width: 25%; }
/* Remove the previous clear*/
  .grid-1-4:nth-of-type(2n+1) { clear: none; }
/* Clear the float every 4th element AFTER the first. This targets the 5th, 9th... in the grid.*/
  .grid-1-4:nth-of-type(4n+1) { clear: left; }
}
```

Wave goodbye to using `position` and `float` for your layouts. While they have served us well to date, their use for layout has been a necessary hack. We've now got two new kids on the block that will help fix all our layout woes – Flexbox and Grids.

Flexbox is great for individual modules, controlling the layout of pieces of content within each of the modules. There are layouts we attempt to deliver that can be more easily achieved using Flexbox, and this is even more true with responsive sites. For more on this, check out CSS Tricks' guide to Flexbox (*netm.ag/flexbox1-26*) or the Flexbox Polyfill (*flexiejs.com*).

CSS grid layout

Grid is more for the macro level layout. The Grid layout module gives you a great way to describe your layout within your CSS. While it's still in the draft stage at the moment, I recommend this article on the CSS Grid layout by Rachel Andrew (*netm.ag/grid-260*).

FINALLY

These are just a few tips to extend your responsive practice. When approaching any new responsive work, take a step back and ensure that you get the basics right. Start with your content, HTML and layer improved experiences upon them and there won't be any limit to where you can take your designs.

Thanks

With thanks to Jeremy Keith, John Allsopp, Stephen Hay, Aaron Gustafson, Ethan Marcotte, Brad Frost, Patrick Hamann, Jordan Moore and Scott Jehl, who all helped form the thoughts in this article.

Resources

PERFORMANCE

- Network Link throttling (*netm.ag/stuntbox-260*)
- Chrome Dev Tools: Canary (*netm.ag/canary-260*)
- Enhance.js (*netm.ag/enhance-260*)
- Github South Street (*netm.ag/southstreet-260*)

POLYFILLS

- Respond.js (*netm.ag/respond-260*)
- Picture.js (*netm.ag/fill-241*)
- Flexbox Polyfill (*flexiejs.com*)

PLUGINS

- TableSaw (*netm.ag/tablesaw-260*)
- Fit Vids (*fitvidsjs.com*)
- Fit Text (*fittextjs.com*)
- Adaptive UIs (*netm.ag/allsopp-260*)
- Label Mask (*netm.ag/label-260*)

ALL THINGS RWD

- Responsive Design Weekly (*netm.ag/newsletter-260*)
- This is Responsive (netm.ag/thisis-260)
- Responsive Design.is (*responsivedesign.is*)

AUTHOR
MARK LLOBRERA
Web designer and developer Mark (@dirtystylus) is currently a senior developer at digital agency Bluecadet (*bluecadet.com*). He blogs at *dirtystylus.com*

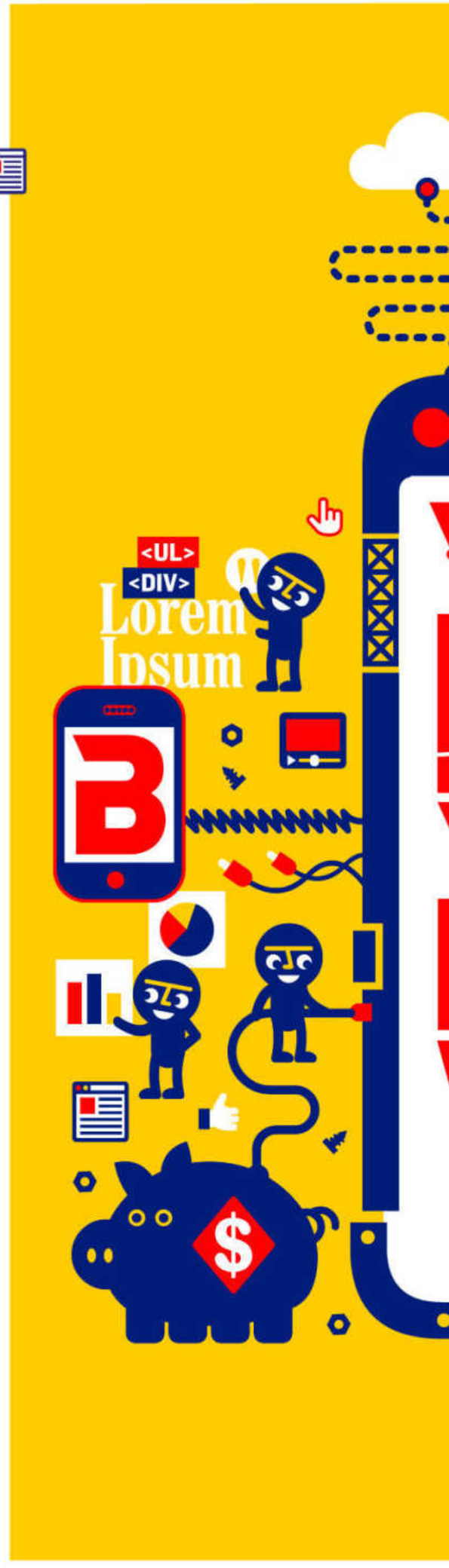
ILLUSTRATION
JAN KALLWEJT
Jan is an illustrator and graphic designer, based between Barcelona and his native Warsaw. He works for clients all over the world
kallwejt.com

Mark Llobrera on how content, communication and collaboration will help you make your next site the best one yet

This article's title aside, there is no perfect site, workflow or tool. But you can probably relate to the feeling that I have at the start of every new project: no matter how well the last one went, I want this one to be even better. To put together this piece I interviewed a group of writers, editors, designers and developers to get an idea of how they go about planning sites. Their answers surprised me.

Instead of a list of favourite techniques and tools (don't worry, there's plenty of those in the resources section!), a few common themes emerged: the importance of content, internal and external communication, prototyping, and breaking down the divide between design and development. Their responses indicated that teams and clients are struggling with bigger, more fundamental issues than which CSS preprocessor to use.

So this is a snapshot of contemporary web workflow, touching on familiar phases and disciplines – content strategy, information architecture, design and development – but placing those in the context of the broader themes just mentioned. There's so much that goes into planning and building a site that it's easy to feel overwhelmed. I hope this article will provide some new approaches to those



www
BUILD
THE
PERFECT
WEBSITE
<UL>
<DIV>
<DIV>
<HEAD>
<TITLE>
<HTML>
<FORM>
<TITLE>
<UL>
<DIV>
<DIV>
<BODY>
<HTML>
<UL>
<DIV>
:DIV

LET CONTENT SHAPE YOUR DESIGN

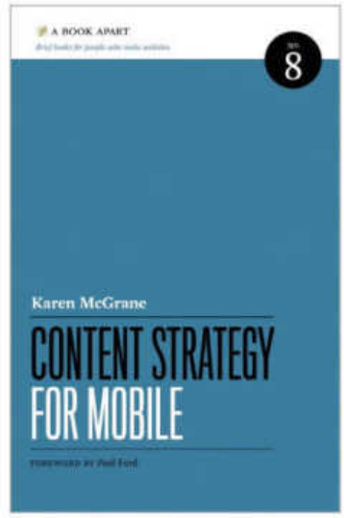
Web designer Stephen Hay shares his insights on content

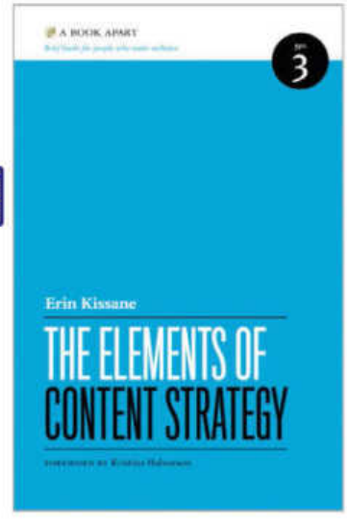
I can't stress enough how important it is, especially when doing web-based design comps, to have *real* content as early as possible in the design process. It doesn't have to be the actual copy that will be used in the final result, but it needs to be representative of the content that is to come.

The idea is to know as much as you can about the content, so you can avoid surprises. In a recent project, when the small number of navigation items suddenly went from three to seven, we had to rethink navigation entirely – keeping it at the top of the page on small screens, and placed horizontally on wider screens was no longer a feasible option. It's a simple example, but a lot of these little issues can be avoided when you get content first.

Another advantage to having content early is that you get to know the 'shape' of the content and can design appropriately. Are there five buttons in this app or 10? Are these things we could use icons for, or are we using text? These choices greatly impact your UI, and you can only account for them when you have content from the very beginning.

Some people don't like thinking about content first, preferring that the designer create a pretty container into which they can pour their content. But that's decoration, not design. Knowing your content is thinking before doing. It's a smart way to design, and it will save everyone a lot of headaches.





Reading list There are plenty of great books exploring content strategy techniques

broader issues, while also giving you new tools to test out and explore.

CONTENT IS THE KINGDOM

At Bluecadet (*bluecadet.com*), the digital agency where I work, my colleagues and I like to start at the very beginning: with the content itself. We focus there, before any devices and their related assumptions are even in play. It's hard, messy work, but how you conduct your research, content strategy and information architecture sets the stage for the steps to come.

Website projects can often be derailed by complex relationships and dependencies that extend past the website you're planning. Content strategist and A List Apart editor-in-chief Sara Wachter-Boettcher (*sarawb.com*) points out we're not simply dealing with a website any more: "The problem is that we've moved far, far beyond the idea of a website as a contained, static piece of a business. The complexity and size and interconnectedness of most organisations' web footprint – sites, apps, intranets, web-based systems and so on – is such that the web affects the jobs of everyone in an organisation. The systems we design need more ongoing management than ever before."

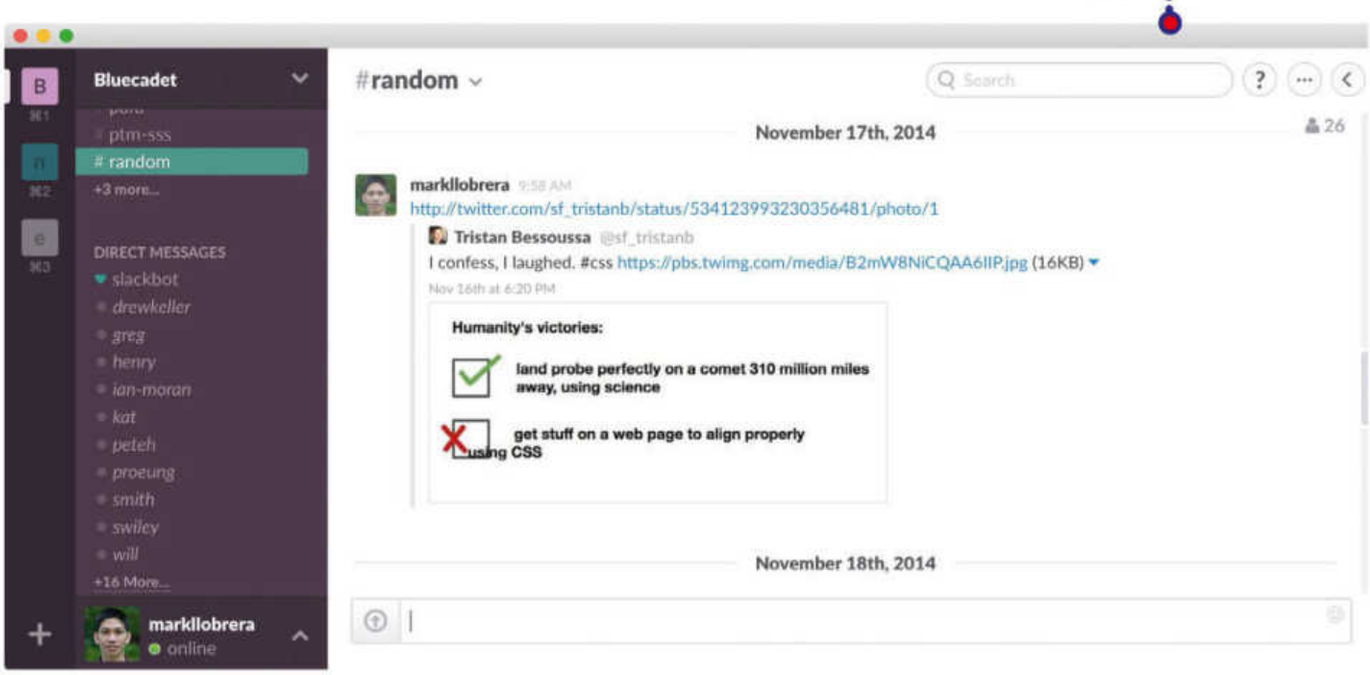
Having this in mind when you conduct your user research and content strategy will help you identify potential risks up-front, while you're still defining the scope of the project.

Content strategy

When you're evaluating content, ask yourself: Is this content useful? Who is this for? How often will it change? Will it vary in format or length?

Determine how the content is going to be produced. Does your client have someone dedicated to creating and maintaining content? Talk to them. Ask them how they work. You want to design and build something that they will be comfortable using, long after you're gone.

Author, editor and content strategist Nicole Fenton (*nicolefenton.com*) echoed this concern when I asked her about common pain points she's encountered



It's good to talk Communication platform Slack (*slack.com*) helps Bluecadet keep its project teams talking regularly

with her clients. She wrote: "I see a lot of companies waste money on software when they haven't taken the time to define their business goals, outline their editorial process or train writers, designers and engineers."

I've found that many design or development problems can be traced back to content problems that weren't properly resolved. On Bluecadet's recent project for *Lapham's Quarterly* (see case study, page 19) we had to restructure two content types in the CMS because we misinterpreted how content authors would associate contributors with their articles. But because we got feedback during our content strategy phase, we were able to redesign those content types and avoid costly refactoring later.

PLAN YOUR COMMUNICATION

Communication was another key topic that came up repeatedly. Almost every single person I spoke to stressed the need to establish good internal communication patterns and reinforce them as a team.

What do some of those patterns look like? Allen Tan (*tanmade.com*), a designer for *The New York Times*, enumerated a few of the steps his team has taken to improve communication: "More meetings for consensus-building. Daily design reviews. More pairing with developers – they don't necessarily need to always sit together, but they should have regular in-person contact."

Tan also talked about using different modes of communication, depending on the situation: "There's built-in time in the process to make visual and functional adjustments [once a dev has implemented a design]. But sometimes it's easier to make the tweaks myself and then send a pull request."

ESTABLISH A HEALTHY RHYTHM OF COMMUNICATION WITH YOUR CLIENT AND YOUR TEAM

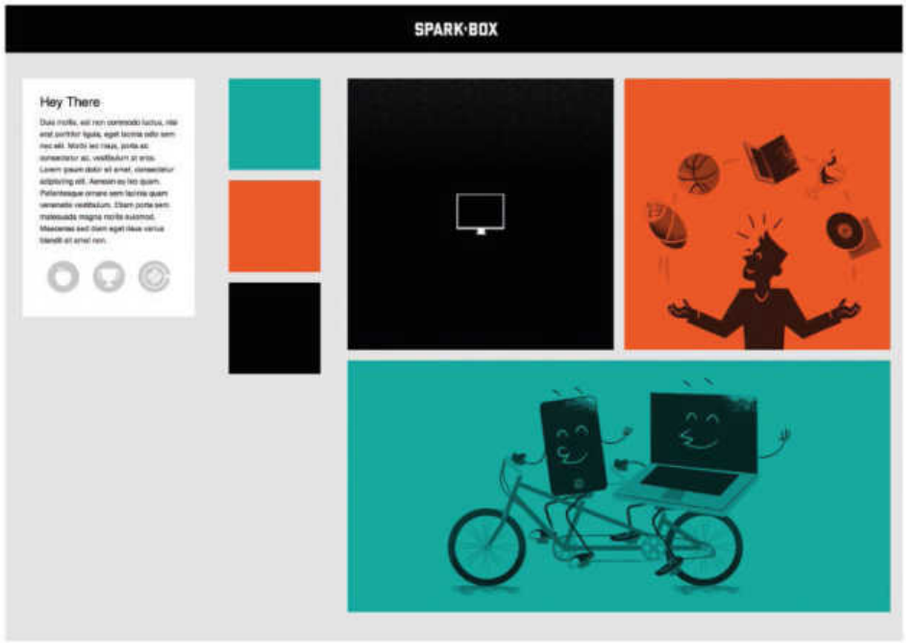
The flexibility that Tan describes can be incredibly useful and liberating for a team. Sometimes it's necessary to meet with a teammate and work things out together, and sometimes it's better for someone to make the adjustment, and then the team can regroup if necessary. Examine your communication paths to see if they are too rigid, or if they introduce so much friction that communication stops outright.

Client conversations

It's crucial to establish clear expectations and lines of communication with your client. At the outset of new Bluecadet projects, we take the time to explicitly state how we're going to work and what the client can expect in terms of communication and artefacts.

Even if we're working with a tech-savvy or prestigious client, we don't assume they know about common artefacts and practices. For each deliverable, we make sure our client understands what that is, and what it's intended to achieve in our design conversations. You'd be surprised at how often the purpose of a deliverable is misunderstood. In addition to the type of deliverables, we also establish their scope ('for this project we will produce two rounds of wireframes'). We also like to clarify what types of client feedback are most useful during our conversations.

Just as you would do with your team, establish a healthy rhythm of communication with your client. The more frequently they are involved, the more you're able to avoid surprises, misunderstandings and lost hours.



Style guides Sparkbox has shared its style prototype on Github: *sparkbox.github.io/style-prototype*

Modular design examples Style Tile (above, *styletil.es*) and Element Collage (opposite, from *rif.superfriend.ly*)

Designer, author and ARTIFACT conference founder Jennifer Robbins (*artifactconf.com*) talked about some of the client communication lessons that came out of the last two years of ARTIFACT: "I think there's a shift towards more frequent communication and more specific communication, [especially] when you're doing the prototyping earlier on … a lot of the decisions that used to be made in a row are now being made concurrently so it just requires a different speed of communication. It just seems like more touchpoints with the client."

PROTOTYPES

Another major theme was the emphasis on prototyping as an important tool.

Under the hood Bluecadet's frontend style guide, including a table of contents and sections of rendered markup

I asked designer and author Ethan Marcotte how his workflow has changed in the years since his book *Responsive Web Design* (*netm.ag/RWD-264*) was published, and he replied: "In my practice, the importance of PSDs and comps has been lessening. They're still incredibly valuable, but prototypes – even rough ones – are becoming more important to early discussions around content, design and functionality."

Marcotte also pointed out how prototypes can help set expectations for your website as a continuum of experiences for different browsers and devices: "I try to incorporate devices as early as possible in design reviews. It does a great job of reinforcing that there's no canonical, 'true' version of the design."

A few designers have created modular, component-based deliverables to replace (or supplement) the full-page comp. Style Tiles (*styletil.es*) and Element Collages (*netm.ag/collages-241*) are two notable ways of giving detailed components (for example: a callout block, a navigation button, a headline and paragraph) enough context so they can be evaluated.

These approaches focus the design conversation on the system or element level, as opposed to a static, fixed-width page. This can lend itself well to the responsive conversation, because responsive sites are fluid, instead of being locked to a definitive size.

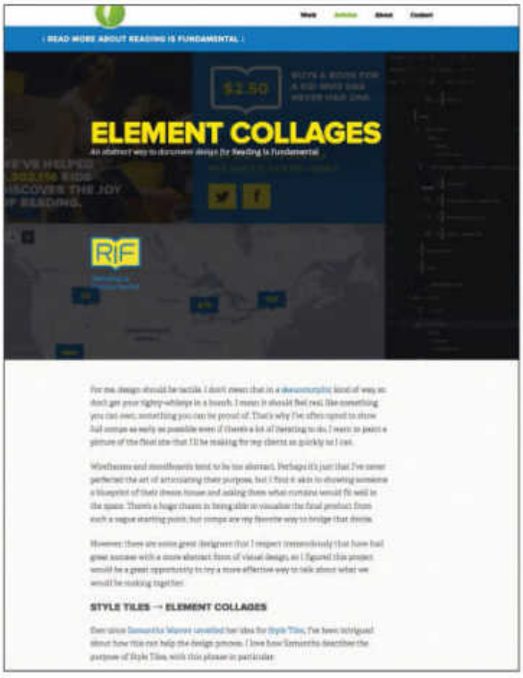
Even with these systems, however, a client will often expect to see a full-page comp, and you may have to decide whether your team is more comfortable showing them that via a static comp or through a prototype.

IT'S TIME TO START PUNCHING HOLES IN THAT WALL BETWEEN DESIGNERS AND DEVELOPERS

DESIGN MEETS DEVELOPMENT

On many teams there's a hard division between design and development. It's time to start punching holes in that wall, and getting the teams to dance. The focus of your team should be on amplifying each other's strengths, as opposed to getting people to do everything well.

At Bluecadet we often run static design iteration in a parallel track with frontend style guide development, pairing a



designer with a developer. We usually create these style guides as a single page directly in the target CMS of our choice, but we've also experimented with tools such as Pattern Lab (*patternlab.io*).

We rely on quick iteration in static comps to make sure we're headed in the right direction, and then we test those assumptions in a responsive context using a frontend style guide. The style guide will reveal limitations in the design, and we can decide whether to iterate in Photoshop or just adjust things directly in markup and CSS.

A bonus to this design pairing? It demystifies the work on both sides. Designers can see how editing CSS can change styles across the entire site, and developers can understand the specific decisions behind a design.

SITE SEAWORTHINESS

One of the tricky things to do is to balance testing with development, especially if you are building your website using progressive enhancement. This means making decisions about which browsers and devices receive a baseline – but content-complete – experience, and which get optimised to take advantage of the latest CSS3 and JavaScript features.

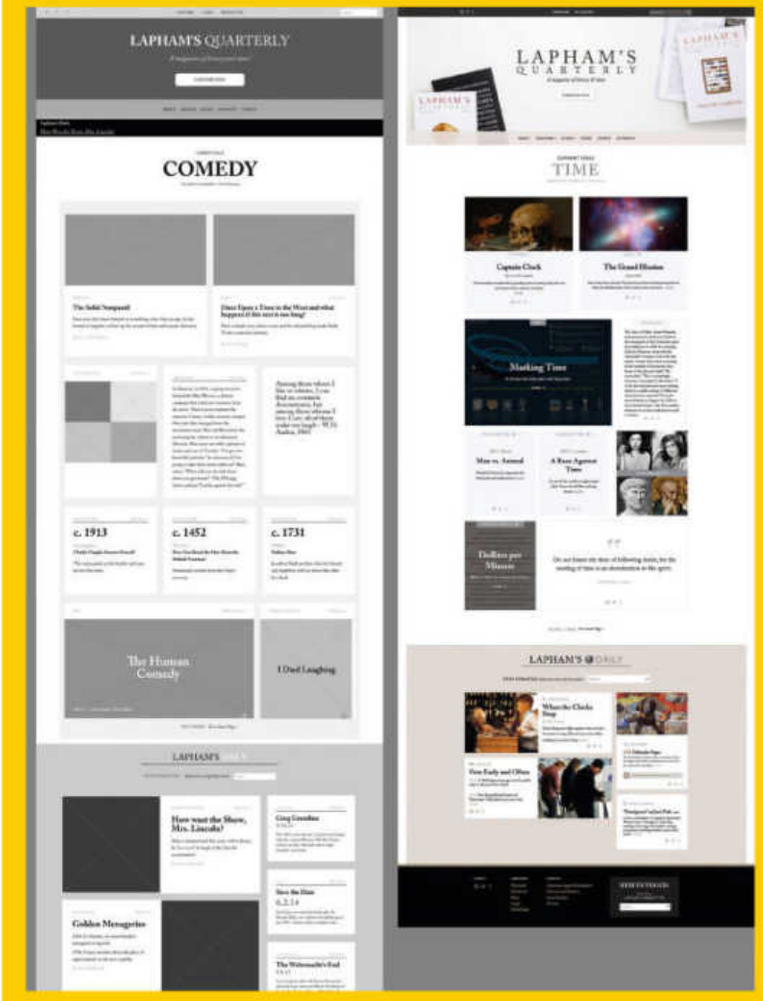
CASE STUDY LAPHAM'S QUARTERLY

Bluecadet was chosen to redesign *Lapham's Quarterly* (*laphamsquarterly.org*), a magazine of history and ideas. Our challenge was how to make the beautiful, 200-page magazine work just as well on the multi-device web.

For this project, we paired a developer with a designer, from creating the very first wireframes through to the design stage, to help identify layouts and modules that might be problematic. As our designer was iterating through layout options, our dev would create quick HTML prototypes for those layouts. We would then test the prototype on a range of devices and discuss the next iteration of design.

The prototypes ended up being strictly an internal deliverable, but they gave us confidence that the static comps we were showing the client were grounded in what we could do in the browser. As we moved further into development, our designer and developer shifted to making design tweaks directly in the browser.

From the beginning to the end of this project, the collaborative workflow freed us to use static tools for quick exploration, and prototypes for adjustments and discussions. The result was a site that translated the robust, sophisticated magazine into an equally compelling online experience.



Making progress A HTML prototype (left), and the final site (right)

RESOURCES

USER RESEARCH

'Why user research is everybody's job' by Emma Boulton (*netm.ag/boulton-264*)
'The pro guide to user research' by Alastair Campbell (*netm.ag/campbell-264*)
***Just Enough Research* by Erika Hall** (*netm.ag/hall-264*)

CONTENT STRATEGY

***The Elements of Content Strategy* by Erin Kissane** (*netm.ag/kissane-264*)
***Content Strategy for Mobile* by Karen McGrane** (*netm.ag/mcgrane-264*)
***Content Everywhere* by Sarah Wachter-Boettcher** (*netm.ag/wb-264*)

COMMUNICATION

InVision (*invisionapp.com*) – a good way to quickly share and annotate comps
Slack (*slack.com*) – feels like IRC on power-ups

DEVELOPMENT TOOLS

Quick JavaScript Switcher (*netm.ag/switcher-264*) – Chrome plugin to toggle JavaScript on and off
Vagrant (*vagrantup.com*) – virtual machines to match server environments
You Might Not Need jQuery (*youmightnotneedjquery.com*) – there's times when you simply don't need jQuery
MicroJS (*microjs.com*) – check this for alternatives to jQuery, or try vanilla JavaScript
GreenSock (*greensock.com*) – I use GreenSock's JS animation platform for complicated animation sequencing

STYLE GUIDES

Styleguides.io (*styleguides.io*) – a collection of frontend style guide resources
Sparkbox (*building.seesparkbox.com*) – a collection of posts on how Sparkbox built its site

PERFORMANCE AND TESTING

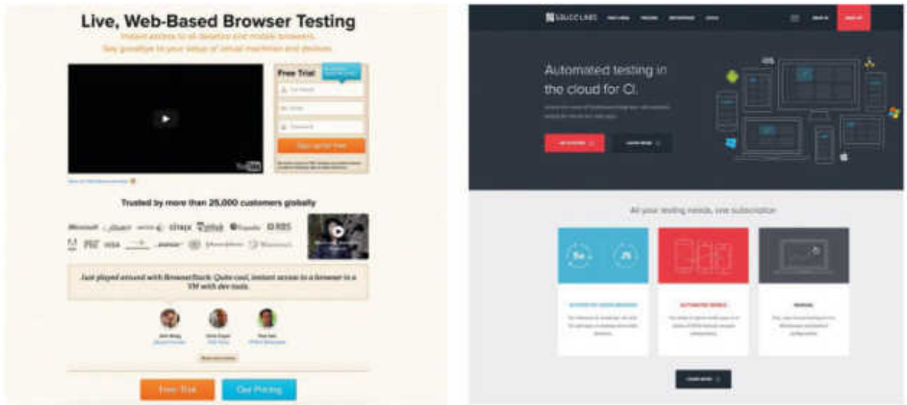
***Responsible Responsive Design* by Scott Jehl** (*netm.ag/jehl-263*)
grunt-perfbudget (*netm.ag/perf-264*) – to test incremental builds against WebPagetest
Firefox archive (*netm.ag/firefox-264*) – an archive of older versions of Firefox
Opera Mini (*netm.ag/opera-264*) – download Opera Mini on your computer
Microsoft's Modern.ie (*modern.ie*) – virtual machines of OS/Internet Explorer combinations

Design meets development Pattern Lab (*patternlab.io*) is a tool for creating frontend design systems

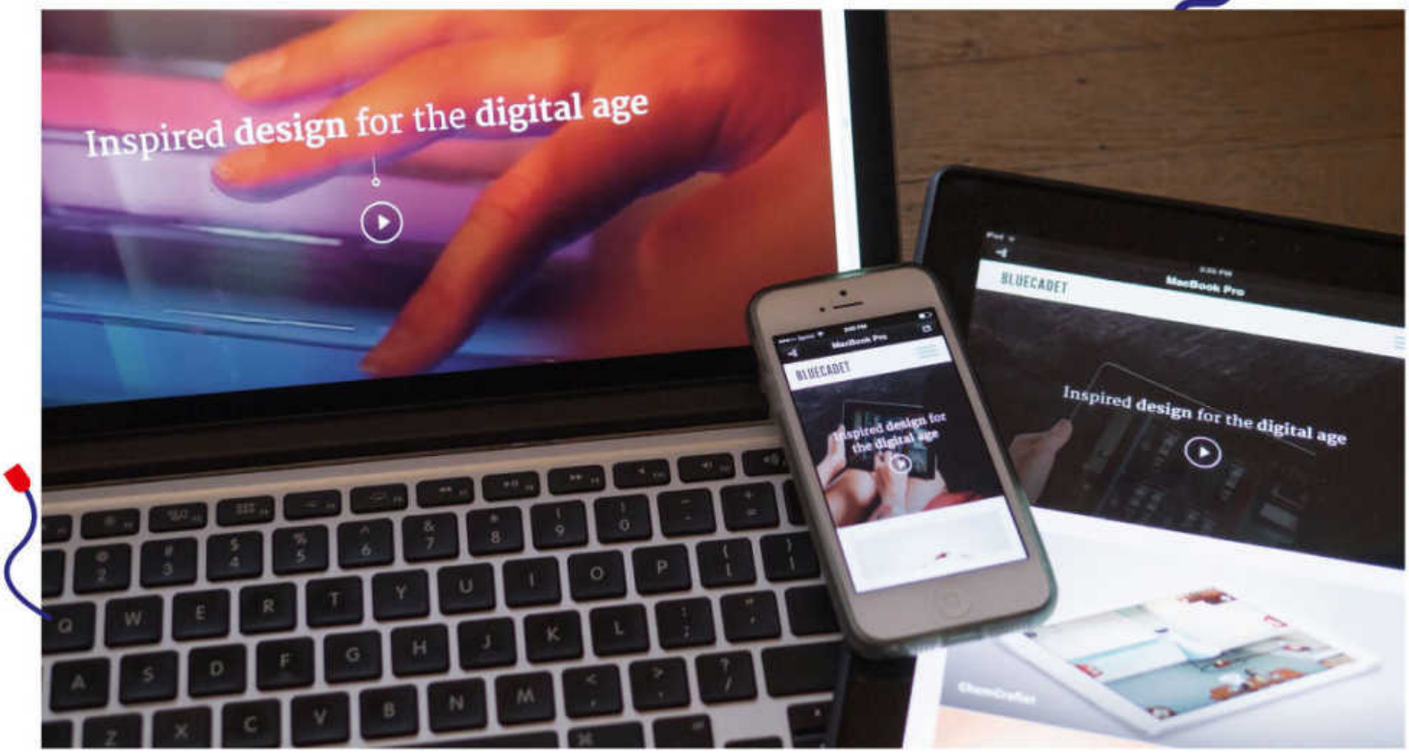
My advice is to make device testing a part of your regular team meetings. My process usually involves testing on a few common devices during sprints, then expanding to the full suite during weekly meetings. The more frequently you test, the fewer hours you'll have to put in later.

For testing your responsive designs, there's no substitute for using real devices. Adobe Edge Inspect (*creative.adobe.com/products/inspect*) allows you to synchronise devices to a master browser, so that you can check multiple devices with ease. We keep a range of devices on hand at the Bluecadet office (older Android 2.2 phones, Android tablets, and varying iOS devices running iOS 6–8). If you don't have a collection of devices for testing, OpenDeviceLab.com (*opendevicelab.com*) can help you find a device lab nearby.

If you're looking for remote testing, BrowserStack (*browserstack.com*) and Sauce Labs (*saucelabs.io*) both give access to a wide cross-section of operating systems and browsers. Testing in this manner can be slow, so I've found it



Remote Testing BrowserStack (left) and Sauce Labs (right) are great resources for remote testing



Device testing Adobe Edge Inspect synchronises multiple devices to a computer browser, so you can quickly check designs and prototypes

more useful for quick rendering checks, as opposed to testing interactivity.

Performance planning

Performance is something that directly affects the bottom line, but as an industry we often treat it as a separate consideration that we do *after* everything else. One of the lessons I've learned is that it's a lot harder to optimise performance if you haven't considered it up-front, so establish a performance budget at the outset.

This could be a limit on the total page weight for any given page of the site (650KB, for example), or a time-driven aim (7s total load time on a 3G connection). Tim Kadlec has been writing about performance budgets for a while on his blog (*netm.ag/kadlek-264*), and Dan Mall recently wrote about how to make one (*netm.ag/budget-264*).

If you've prioritised performance, the next step is to measure and optimise for it. I prefer to use WebPagetest (*webpagetest.org*) and Google's Page Speed Insights (*netm.ag/insights-264*) to diagnose and optimise my sites' performance.

ALL TOGETHER NOW

Researching this article revealed two parallel, complementary threads. On one side, our tools are better and our techniques are adapting to the increasing range of devices and browsers. That's the straightforward part: exert enough brainpower and technology, and you'll eventually reach a solution.

FOR THAT PERFECT SITE, YOU SHOULD PAY EQUAL ATTENTION TO THE DETAILS OUTSIDE YOUR DISCIPLINE

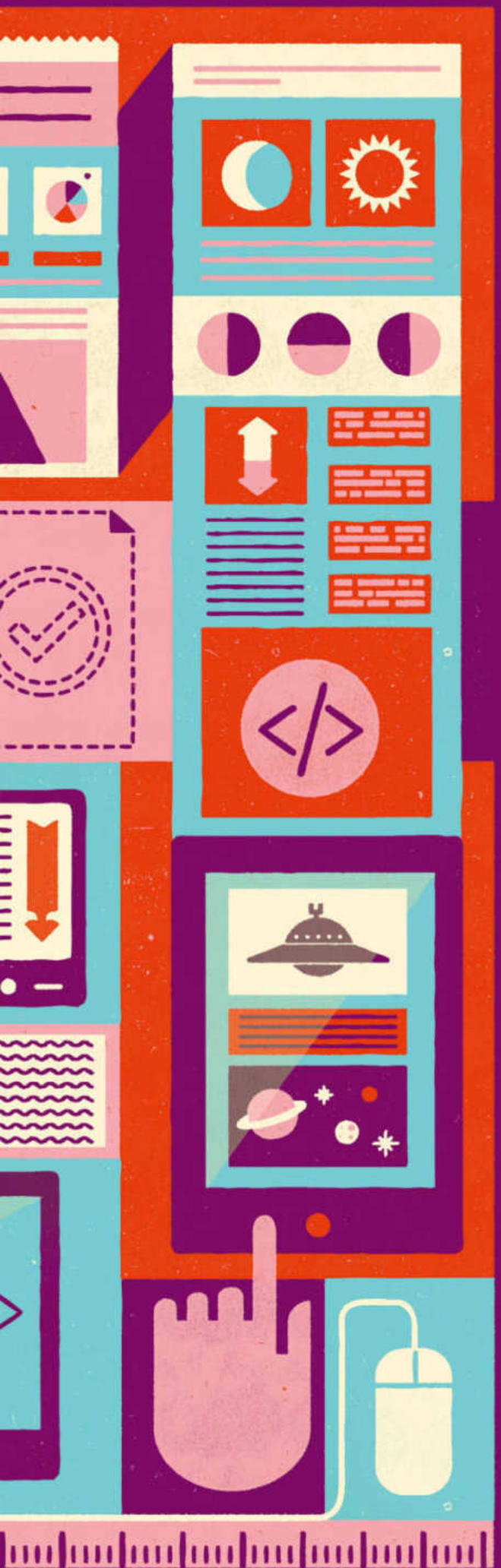
But there's another side, and it's equally important. Creating a site today requires better internal and external communication and collaboration. We still need to face our old, familiar content problems – they haven't gone away, they're just magnified by the multitude of ways in which content can be delivered and consumed.

If you're like me, you might be tempted to lean heavily on that first side, and focus tightly on the fine-grained details of design and development. And that's fine, to an extent – you can actually make a pretty good site by ensuring that the craftsmanship in your discipline is top-notch. But I think if you're really going for it – if you're reaching for that elusive, perfect site – you should pay equal (if not more) attention to the details outside your discipline.

That might involve putting yourself in your teammates' and clients' shoes more frequently, or tweaking your team's tried-and-true workflow to make it better. And if that sounds like a call for more empathy in design, that's exactly my point.

Finally, sharing your experiences is absolutely essential – the web does not stand still, and we depend on each other to push and pull ourselves along with it. Finding and sharing the good and bad parts of your process helps everyone, because our community has shown time and again that we can solve problems – and make better sites – together.


CREATE
THE PERFECT
LAYOUT
AAA



There are a number of ways to create responsive layouts without media queries. **Vasilis van Gemert** shares some useful techniques, ranging from primitive to highly powerful

Thanks to media queries, we can create websites that work on any device. Without them the current web simply couldn't work. But there are other options as well. In certain situations, a different technique can solve our responsive design problems in a more systematic, algorithmic way. If you are, like me, a lazy person, you might be interested in designing these kinds of layout systems whenever possible. Systems like these are self-governing – they have to be able to make decisions for themselves and for us, based on the constraints we give them.

We've always been able to create flexible layout systems on the web. Even with ancient CSS properties like `float` you could create layouts that adapted to different screen sizes. Recent implementations of new CSS features, like flexbox and viewport-relative units, have enabled us to take CSS layout to a level of flexibility we could only dream of before.

AUTHORS

VASILIS VAN GEMERT

Vasilis (*@vasilis*) is a lecturer at the Amsterdam University of Applied Sciences. When he's not teaching web design, he's probably in his hammock, creating art for the web

ILLUSTRATION

BOBBY EVANS

Bobby is an illustrator based in north London, who has worked with clients as varied as Penguin, Habitat and Vodafone. He also runs the award-winning Telegramme Paper Co with Kate Brighouse

telegramme.co.uk

Lists CSS columns are perfectly suited to lists, both short and very long

Viewport boundaries CSS columns only work for articles if the content fits within the boundaries of the viewport

Inline-block By using float or inline-block, the layout will change by itself if the screen is big enough

Horizontal scroll CSS columns might work for articles if we can convince the user to scroll horizontally

Float and display: inline-block

Let's start this exploration of layout systems with a simple example. Many blogs show a list of introductions on their homepage. On a small screen you probably want to display these articles one on top of the other, but on wider screens a layout with more than one column makes sense.

If you give each article `max-width: 20em` and tell it to `float: left`, the browser will simply show as many columns as possible, depending on the width of your screen. This has worked since we invented floats. Floated items do not automatically clear; they bump against higher items. This means a blog that has been laid out this way will sometimes show gaps. Depending on how you look at it, the chaos of floating things can either be a cause for concern or a wonderful, serendipitous design feature. You can think of these gaps as being chaotic, or you can call them 'active white space' and be very happy with them. If you don't like white space you can use `display: inline-block;`.

Float is a very simple technique that results in primitive yet flexible responsive layout systems. It works just fine without media queries, although you might want to add a few to tweak the design if needed. Bare-bones floating might not be perfect for all use cases, but it's definitely an option worth considering.

Multi-column

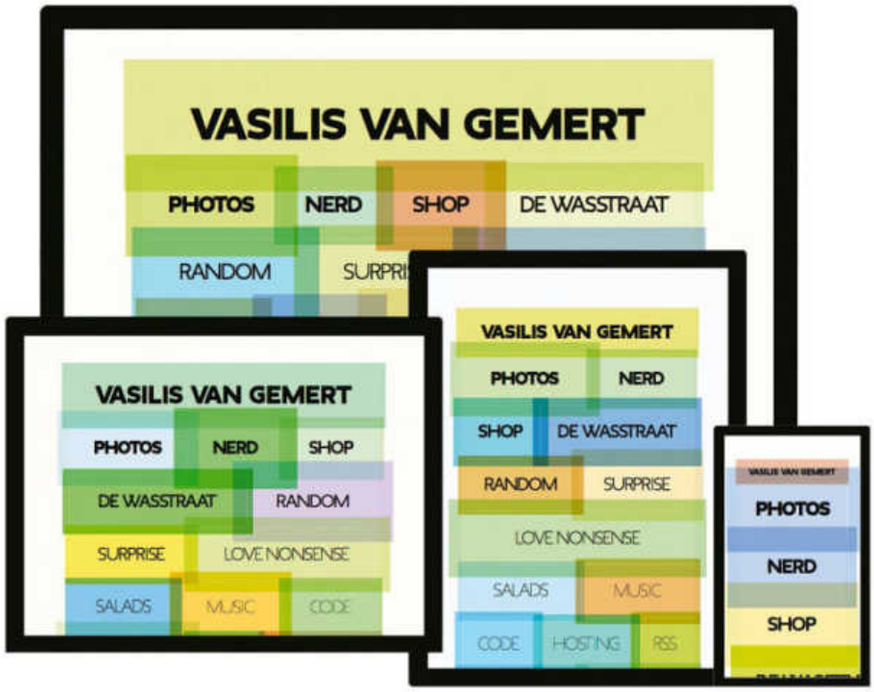
I'm not sure why – perhaps because of buggy and incomplete implementations – but you don't see many websites that use CSS columns. This is a pity, since it's a highly flexible technique. It's perfect for lists of links, like navigations, footers, search results or photo blogs.

CSS columns are almost never used for articles. They become horrible to use when the article is higher than the viewport. People simply don't want to scroll down and up again while reading. There is a very nice and simple way to use columns in articles, but only if you can somehow convince your users to scroll horizontally.

You can create a simple horizontally scrolling multi-column layout by setting the height of the article to a maximum of 100 per cent of the viewport, and by telling it to use columns of no less than 20em. The problem you now need to solve is how to explain to your visitors that they should scroll horizontally.

Float is a very simple technique that results in primitive yet flexible responsive layout systems. Bare-bones floating might not be perfect for all use cases, but it's definitely worth considering

You could add new UI elements to clear this up. Or if you're lazy, you could choose to always make sure the columns never fit completely in the viewport. It's remarkably simple to achieve this. These few lines of CSS are all you need:



Em units Text set with em as a unit will be too big on small screens for this particular design

```
article {
 columns: 20em; /* never be smaller than 20em */
 height: 100vh; /* be as high as the viewport */
 width: 75vw; /* be 75% of the width of the viewport */
}
```

There are many reasons why CSS is awesome, but the fact that the initial value for `overflow` is `visible` is definitely one of them. Without that weird but fantastic CSS property this multi-column layout wouldn't work.

Flexbox and the viewport

Tab Atkins gave a very clear description of what flexbox is:

"Flexbox is for one-dimensional layouts – anything that needs to be laid out in a straight line (or in a broken line, which would be a single straight line if they were joined back together)"

That sounds a bit like `float`, but of course it's much more powerful. With flexbox you can create simple looking layouts that would have been impossible a few years ago. For instance, you can tell items what to do with any leftover white space. You can leave it at one of the ends, you can distribute it evenly between (or around) them, or you can choose to stretch the items – which basically gets rid of the white space.

I used this last option to lay out the list of links to my online activities on my homepage. I didn't want this list to be an orderly multi-column layout, I wanted the browser to simply fit as many items on each row as possible. For the first version of this design I used this code:

```
ul {
 display: flex;
 flex-wrap: wrap;
 font-size: 2.5em;
}
```

This looks OK on large screens, but is problematic on smaller screens (see image above). I wanted all items to always be visible inside the viewport, no matter the size. To fix this, I could have used media queries, of course. But wouldn't it be great if the browser could somehow adapt the font size

GRIDS AND SHAPES

Not all content needs a design system. There are many things – like longer, one of a kind articles, books and magazines – that need to be carefully crafted by hand. To design this kind of content you need fine control over how things are laid out. Many magazine designers have been laughing at the web for ages, since we couldn't use shapes to lay out text around images.

Quite a few browsers have implemented CSS Shapes, which lets you create fantastic, magazine-like layouts. Layouts that use these shapes a lot will probably need media queries to make sure everything works on all screen sizes. But the things we will be able to do with them will be amazing.

More and more browsers are also starting to ship experimental implementations of CSS Grid Layout. This specification lets you define table-like structures you can place your content in. Its true power lies in the fact you can visually rearrange the content in any way you want, while the source order stays the same.

This in combination with media queries will allow you to create websites that present the content in a truly optimised way for any viewport. Content choreography, as Trent Walton calls it, will finally be easy to do.

I'm especially looking forward to seeing the fantastic things we will start to make once we start combining CSS Shapes and CSS Grids with viewport-relative units. My mind is blown already.

Example Here, Emil Björklund uses CSS Grid Layout, CSS Shapes, the clip-path property and more

CSS COLUMNS FOR ARTICLE LAYOUT

On the web we are used to scrolling from the top of the page to the bottom. This is why we never use multi-column layout for long articles. Scrolling to the top of the page is not what you want when you're halfway through reading an article. This is unfortunate, since multi-column layouts are fantastic for long texts. There is a reason why they are so successful in paper magazines and newspapers.

There are two ways to use multi-column layout on the web. A few years ago Håkon Wium Lie of Opera Software proposed a new way of reading long articles (*netm.ag/lie-274*): no scrolling, but 'paging' – you would flip from page to page like you do in books. With this you could create the same kind of layouts that we see in newspapers, only much more flexible. Unfortunately, no browsers were interested.

The other way to use multi-column layout for long articles is by making the article as high as the viewport, and scrolling sideways. The problem here is how to convince your users this is a good idea.

There is a very interesting startup, Blendle (*blendle.com*), that succeeded in convincing its readers. With a Blendle account you can buy and read complete articles from Dutch newspapers and magazines. It uses a multi-column layout to give its site a magazine-like look and feel, and by scrolling sideways it remains usable. The result is remarkably nice to look at and to use.

Blendle This Dutch startup convinced users to scroll horizontally

to the size of the viewport? It turns out that, indeed, you can use viewport-relative units to do exactly that.

Viewport-relative units

Instead of using `em` and media queries to make sure our layout works in different viewports, you can use the `vw` unit, which is a percentage of the width of the viewport. If you set the font size to `2.9vw` (for example) the layout will always fit in the viewport, as you can see in the example pictured below.

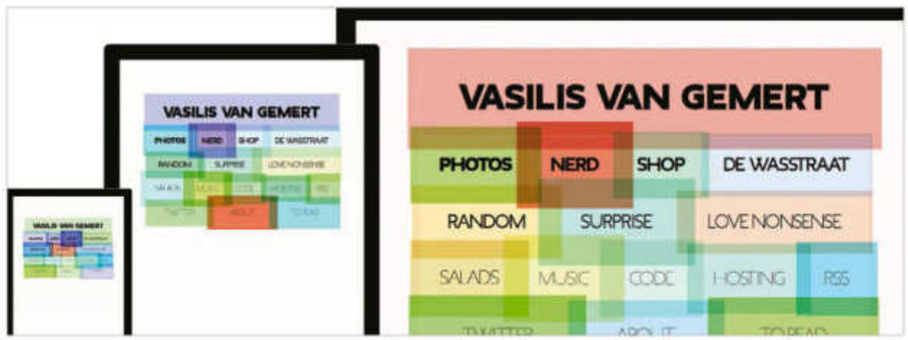
There is a serious usability problem with using the viewport width as a unit for text though: it can easily become too small to read on very small screens. This is an even bigger problem since there is no way for the user to increase the font size. Browser zoom does not work with viewport-relative units!

You can make sure that the font size never becomes too small by using a calc function: `font-size: calc(1em + 1vw)`. This works like a minimum font size. But while this is a clever trick to make sure our text is always accessible, it doesn't solve the problem of how I can make all the links on my homepage fill up any viewport.

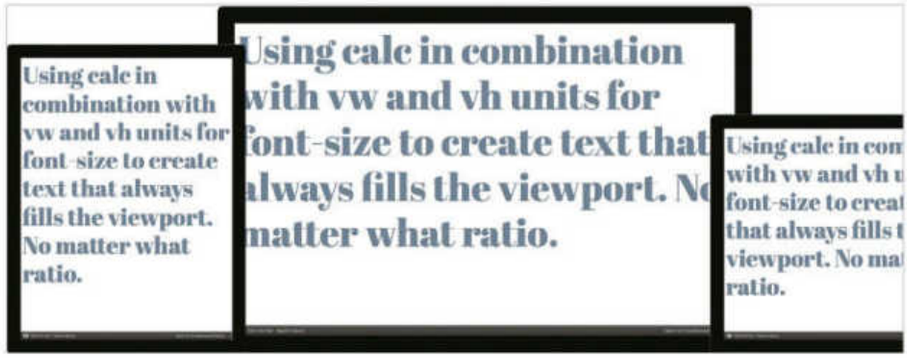
The beauty of these techniques is that you order the browser to behave in a certain manner. Instead of designing every possible layout, you let the browser and the content figure it out

Viewport calculations

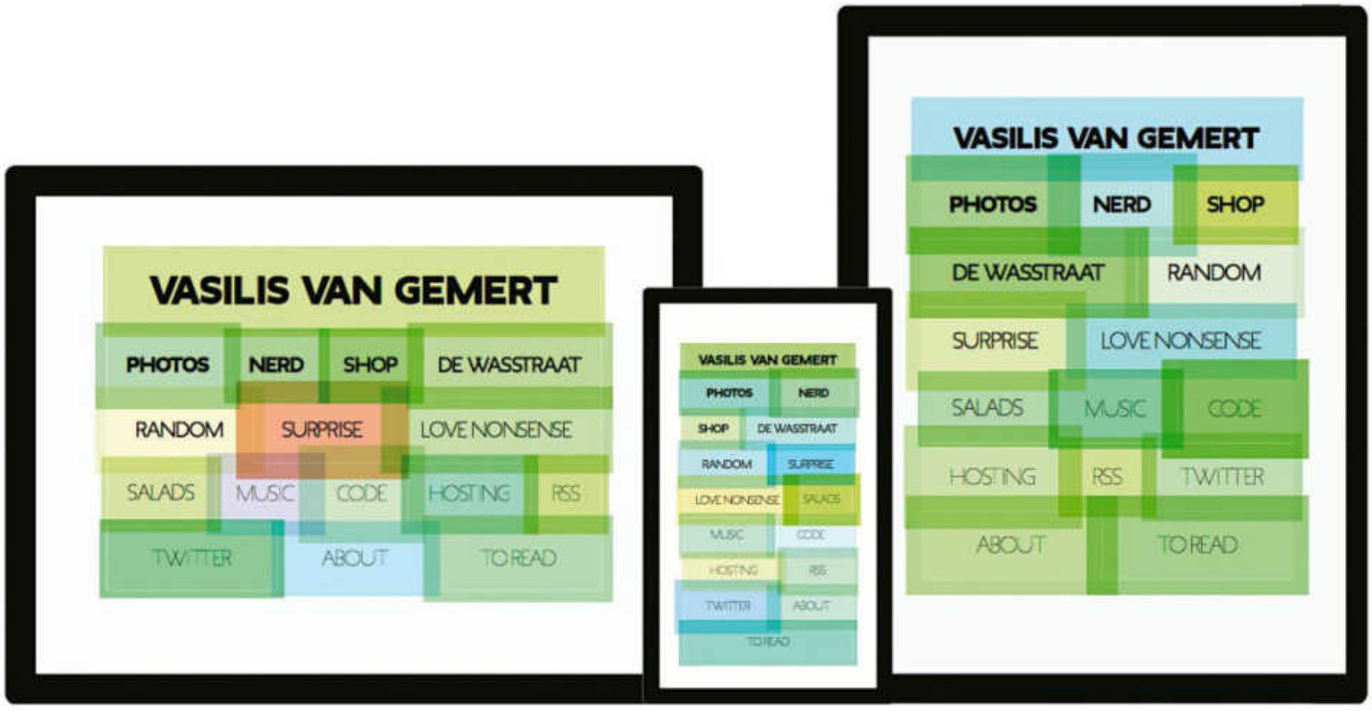
It's not really possible to always perfectly fit any text into any viewport



Resizing content By using vw as a unit, we can resize the content as if it were an image



Calc plus vw If we combine calc with vw and vh units for font size, we can create text that *always* fills the viewport



Calc function If we use a calc function we can make sure the content will fill up the viewport

with just CSS, but we can come close. There's a brilliant CodePen (*netm.ag/voor-274*) where Dillon de Voor explains this line of code:

```
font-size: calc(4vw + 4vh + 2vmin);
```

With this specific calculation, this one sentence, set in a certain font, will always fit in any viewport.

For my own site I changed the numbers in this `calc` function a little bit, and now the layout on my homepage does what I want. The content will always try to fill up the viewport as much as possible.

Quantity selectors

The beauty of these techniques – float, CSS columns, and flexbox in combination with viewport calculations – is that you order the browser to behave in a certain manner. Instead of designing every possible layout for every possible screen size, you let the browser and the content figure it out together.

Of course, this doesn't always work. Often you need finer control over the different layouts, and for extreme screen sizes you need to define exceptions. For these cases media queries are usually the tool we need. And they're fantastic.

Container queries

Soon after we started creating responsive designs with media queries, we found out that, while they are extremely useful, they can't always help us achieve what we want. I bet you've often wished something like 'element queries' existed: changing the way elements look depending on their own size, instead of on the size of the screen. It makes so much sense.

The reason these don't exist yet is because they could cause endless loops. What should the browser do if you tell it that elements wider than 30em should be 20em wide? The Responsive Issues Community Group has come up with a possible future solution: we can't use element queries, but we might be able to use container queries. With a container query you wouldn't be able to style the element itself, you could only style its children. They might look something like this:

```
article:media( min-width: 30em ) screen {
  ...
}
```

The proposed syntax will probably change in the future, but the basic idea makes sense.

Quantity selectors

Aside from screen size, there are other conditions you can use to style things with. There have been a few brilliant articles recently about styling things based on quantity.

It turns out you can apply different styles to an element based on the number of siblings it has. This technique doesn't use media queries or container queries, of course; it makes very clever use of type selectors.

There are some wonderful and very useful things you can do with this technique. Quantity selectors are quite handy for search results, where you don't know if there will be one single result or hundreds of them. You can now change the way they look based on the number of results. It's handy for filtering as well, as you can see in the example

RESOURCES

Quantity Queries
netm.ag/pickering-274
An article written by Heydon Pickering

A Dao Of Web Design
netm.ag/allsopp-274
John Allsopp's all-time classic about the flexibility of the web

Viewport calculations CodePen
netm.ag/voor-274
A brilliant mathematical explanation of viewport calculations by Dillon de Voor

Container Queries
netm.ag/marquis-274
In this, Mat Marquis explains the current state of container queries

Grid by Example
gridbyexample.com
A fantastic site with lots of practical examples about Grid Layout, by Rachel Andrew

CSS Shapes 101
netm.ag/soueidan-274
In this, Sara Soueidan explains in depth what you can do with CSS Shapes

Examples in this article
The best way to learn new techniques is by playing with them. Here's a list of all the examples I used in this article.

- Floating blogs (*netm.ag/floating-274*)
- A multi-column photo blog (*vasilis.nl/voto*)
- Multi-column article (*netm.ag/multi-274*)
- A list of links (*vasilis.nl*)
- Mixing vw, vh and vmin (*netm.ag/vw-274*)
- Filtering with quantity selectors (*netm.ag/quantity-274*)
- Playing with Grid Layout and CSS Shapes (*netm.ag/layout-274*)

Clever design By using a combination of flexbox, quantity selectors, and a few viewport-relative units, clever design systems emerge

Small sizes The size of the images is small, in order to fit all the pictures of the speakers inside the viewport

Make some space When a filter returns fewer results, the individual images take up much more space, of course

above, in which it has been used to enable users to filter all speakers of the Frontend Conference in Zürich.

Initially, the user can see all the speakers and hosts. All images are small so they fit within the viewport. But when they filter the results to only show the hosts, and not the speakers, there's much more room. This brilliant technique gives us the possibility of filling up the viewport with just the two images.

The selectors that make this possible look quite complex at first. At first, I didn't understand how they worked at all. Luckily for us, Heydon Pickering has done a fantastic job in explaining quantity selectors in this article: *netm.ag/quantity-274*.

```
article {
 flex-basis: 100vmax;
}
article:nth-last-of-type(n + 2),
article:nth-last-of-type(n + 2) ~ article {
 flex-basis: 50vmax;
}
article:nth-last-of-type(n + 6),
article:nth-last-of-type(n + 6) ~ article {
 flex-basis: 33.33vmax;
}
```

This block of code tells the browser each `article` should preferably be as wide as the longest side of the viewport. But if there are two or more articles, they should be 50 per cent of the longest side of the viewport. And if there are six or more, they must be 33.33 per cent.

`Vmax` doesn't just fill up the viewport in an efficient way, it also changes the composition depending on the ratio of the viewport. With just a few lines of code you can now create this layout system that responds to the content *and* the viewport.

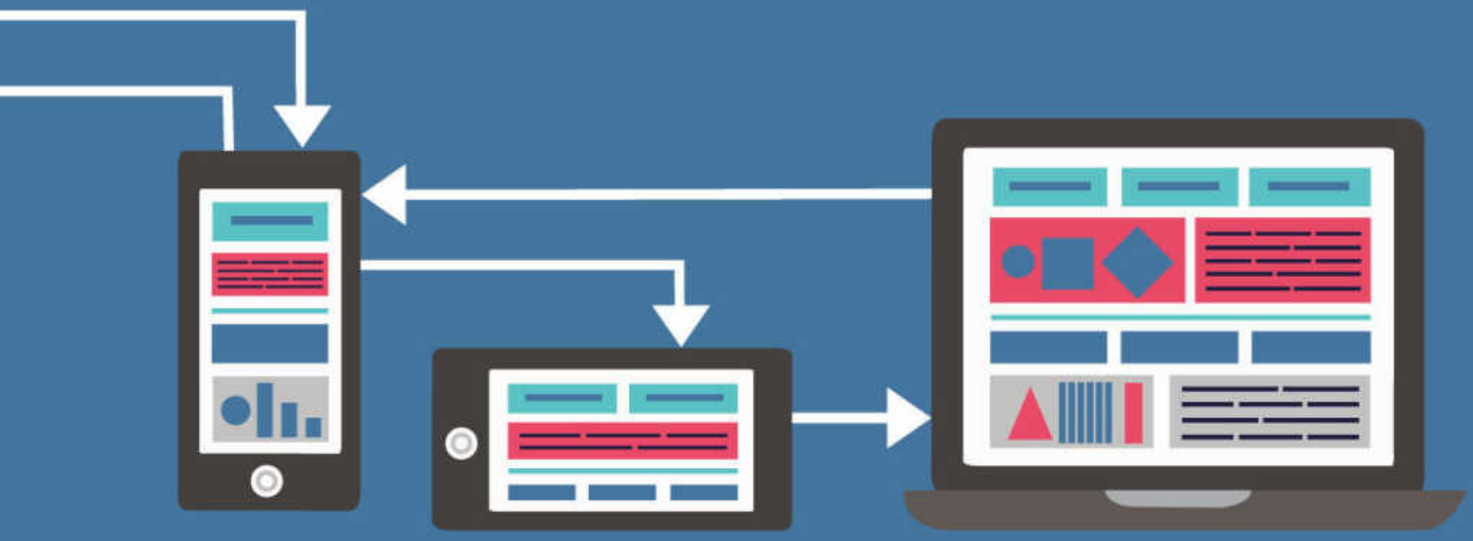
Is this ideal?

We have always been able to let the browser decide what the layout should look like based on a few clear orders: use as many columns as needed, as long as they are not smaller than 20em. Ideal for lazy designers and developers.

New CSS features like flexbox and viewport-relative units have made this way of designing much more powerful. We're only beginning to scratch the surface of these super-flexible, clever layout systems. And the results are already fantastic. I'm really looking forward to seeing the brilliant stuff you come up with once you start exploring all the other possibilities of these CSS techniques.

We have always been able to let the browser decide the layout based on a few orders. CSS features like flexbox and viewport-relative units make this approach much more powerful

Sure, these kinds of design systems can't be used in every single use case. There are other CSS modules – like CSS Grid Layout and CSS Shapes – that give us fine control over our compositions. But whenever possible, we could let go of this control and let the browser and the content figure things out. Or, like John Allsop says in 'A dao of web design': let's "accept the ebb and flow of things".



MULTI-DEVICE DESIGN
BEST PRACTICES AND PITFALLS

Patrick Haney and **Jenna Marino** take you through the practices to follow and mistakes to avoid when designing for smart devices of all shapes and sizes

AUTHORS

PATRICK HANEY

Patrick is not a sausage. He's a designer in Western New York who lives for frontend code, craft beer and Buffalo wings
patrickhaney.com

JENNA MARINO

Jenna is a senior UX designer who helps shape the experience of web, mobile and television applications at Univision
hanerino.com

Designers today have a number of things working in their favour. They are able to search a vast collection of resources for documentation and inspiration online, they can become part of a community around specific devices and operating systems, and they can easily collaborate and share knowledge with their colleagues.

But with all the technological advances in our society, a designer's responsibilities have begun to bleed into other areas as well. Design constraints are no longer just about screen sizes and operating systems. Here are some tips and tricks to help any designer in our new, multi-device, connected world.

STATE THE OBVIOUS

As designers, sometimes we take things for granted. We've become familiar with the notorious hamburger icon as a navigational prompt, but how many of our users are still unaware of what it means? And how can we help ease the transition to these new interface metaphors?

Think about the last time you found yourself in front of a restroom with cleverly renamed signage for 'Men' and 'Women'. Did it take you an extra second or two to get the right door? Ambiguous restroom labels are almost as annoying as the obscure UI patterns we've begun to see in our websites and apps.

Being clever may add delight, but the obvious always wins. As Luke Wroblewski points out, hiding parts of your app behind menus, ambiguous or not, can greatly decrease engagement (*netm.ag/wrobleski-271*). It's a case of out of sight, out of mind.

For smart TVs, it gets even worse. Hiding important information under a menu on a television screen never

TVs: STAY IN CONTROL

Take a look at your TV's remote control. How many of those buttons do you actually use? Probably 10 at most, including the four directional buttons, a selection button, channel up and down, volume up and down, and the power button. When designing applications for TV, be aware that each brand has its own design and button arrangement, not to mention special hardware buttons. And these remotes differ, even within a single brand's lineup.

Considering all those buttons, you'd probably expect a lot of options for TV app developers. Not so fast. The only inputs you can rely on when designing for TV are the four directional buttons and the remote's selection button.

When it comes to TV interfaces, the directional buttons are your friends, because the less you force the user to look down at the remote, the better. A TV-watcher does not want to deal with complicated tasks, but instead is expecting to leisurely browse or quickly find the program they're looking for.

And don't forget about voice and gesture inputs. Many smart TVs now include a microphone and/or camera to track your movements and listen to commands. This brings a new set of input methods to consider. With gesture, you'll need to take into account variance – i.e. the shakiness of a person's hand while trying to select an item, and how long it has to be held up in the air to navigate.

With voice commands, the on-screen interface must display button labels for voice controls, including accents and languages, for both accessibility and learning curve improvements. Microsoft, for instance, limits the number of voice commands on Xbox apps. Once again, the aim is to be as simple and concise as possible in your design.

Hamburger issues Rather than choose between the hamburger icon and the label 'MENU', nGen Works combined the two

works out for your users. They typically don't know to look there – and even when they do, menus are difficult to navigate using traditional television input methods.

Instead, make sure your interface elements are front and centre, and their intent is obvious. Use straightforward labels, with icons to reinforce their meaning. The folks at nGen Works integrated the hamburger icon into the navigation on their site by replacing the 'E' in the 'Menu' label (*netm.ag/ngen-272*). Obvious and clever: the perfect combination.

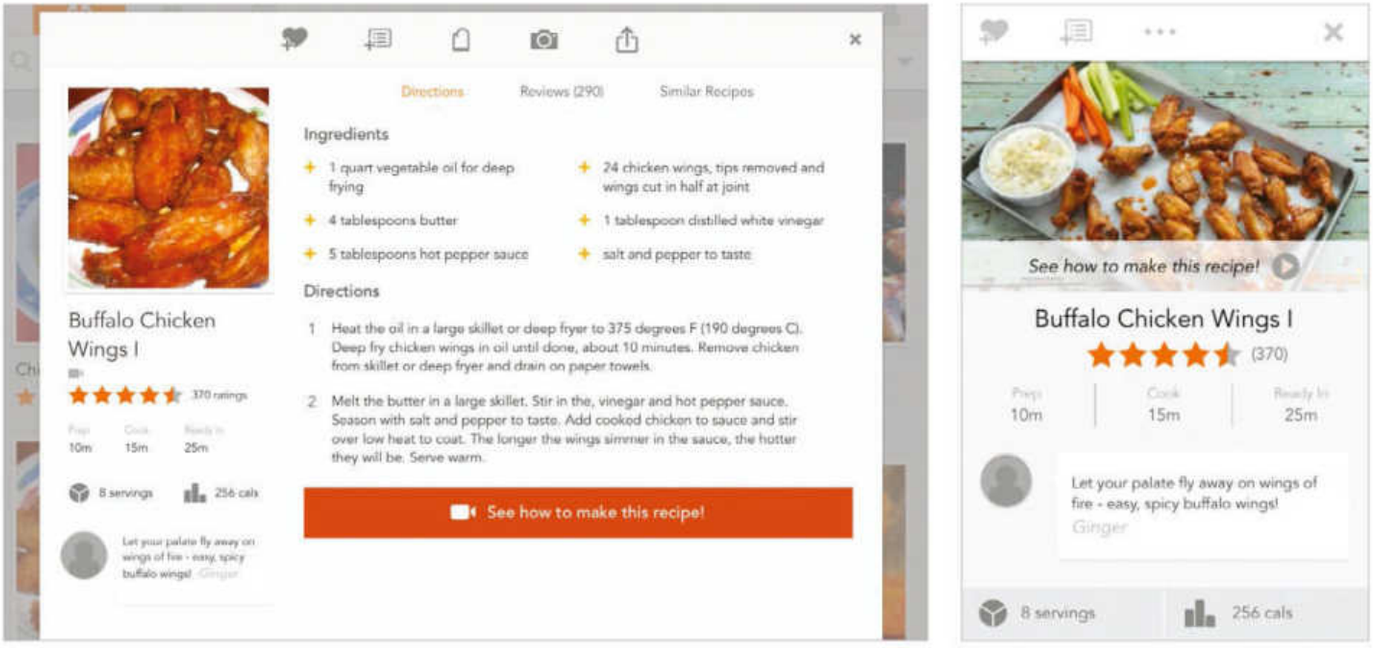
In fact, using the word 'Menu' rather than the hamburger icon can increase engagement by up to 20 per cent, according to New Zealand-based web developer James Foster (*netm.ag/menu-272*). These familiar three bars may not be as familiar as we think.

VALUE USERS' INPUT

The way that we interact with our computing devices is constantly changing. Keyboards, mouses and trackpads are commonplace and will be around for some time to come, but touch and multi-touch interfaces are now also widespread. So much so that our one-year-old already knows how to use our phones and tablet. With 350,000 new 'users' born every day, best practices for touchscreen interfaces should be a concern for every designer.

With multi-touch interfaces, the most common gestures are scroll, swipe and tap. Pinch to zoom is also commonly understood, but if it means zooming in and scrolling left and right, most people will tend to abandon what they're doing and look for another way to browse the site.

Best in class The Netflix brand and user interface translates well across numerous devices



Design shifts The AllRecipes app for iPhone and iPad both take advantage of their respective screen sizes

Touchpoints are also critical to your application's interface, as finger sizes range from toddler to hulk. Apple recommends a minimum 44 x 44px touch size to optimise the ease of tapping an interface element. Microsoft and Nokia recommend similar sizes in millimeters: 9x9 and 7x7 respectively.

But keep in mind that our applications are moving outside of the desktop and mobile realm and into our TVs, through set-top boxes and native smart TV apps, where the remote becomes the most common input device. And that can be a scary thought for designers.

FOLLOW THE THREE Cs

In the book *Designing Multi-Device Experiences* (*netm.ag/levin-272*) author Mihal Levin discusses what she calls the three Cs of design. She argues that consistent, continuous and complementary experiences are the key to dealing with design across a multitude of devices and screens.

Be consistent

Consistent design is important in two areas: the interface and the brand. Do I recognise the app when I switch devices, and do I feel confident I can pick up where I left off? When you use Netflix on your Apple TV and need to leave the house, can you easily find your way around the iPhone app? Netflix has done an incredible job of making sure its brand, interface elements and hierarchy of information translate well across the numerous devices it supports.

> When in doubt, follow expected device patterns and avoid breaking existing, learned behaviours

Speaking of TV, the screen layout is integral to helping your users accomplish their goals easily. Using consistent templates for TV applications reduces the learning curve considerably. Stick with two or three layout templates at the most, and make your navigational methods predictable.

But we can't all be like Netflix. When in doubt, follow expected device patterns and avoid breaking existing, learned behaviours. Android devices differ greatly from iOS devices, and their users should have a preconceived idea of what to expect from each. Consistency is especially important, considering 90 per cent of people use multiple devices in sequence to accomplish a task.

Continue the journey

Cooking a meal at home can be broken down into a few obvious tasks, including searching for a recipe, buying the ingredients and following directions. Let's map those tasks to common devices: we might use a computer at our desk to find and save the recipe, our smartphone to check off ingredients at the grocery store, and our tablet to watch prep videos and follow steps in the kitchen.

Remember that not every device is good at, or typically used for, each action required to accomplish a goal. It makes sense to highlight the tasks that are more likely to be done in each context, rather than making everything work exactly the same everywhere. Optimise the experience within the context of the task and play to the strengths of the device.

Keep it complementary

As we all know, our devices are not always used sequentially, as they were

TEST ON REAL DEVICES

Application development has got easier thanks to fantastic simulation tools, but there is no substitute for testing on an actual device. The experience of using a smartwatch app on your wrist, or navigating a television app with a remote control in your living room cannot be fully replicated using a hardware simulator on a computer.

Early Apple Watch development is a perfect example. Developers were creating applications for a device that didn't even exist, and using a simple simulator that provided a rectangle for their app to run in. But that rectangle didn't replicate the black bezel around every Apple Watch that acts as a buffer between interface elements and the edge of the device.

Without it, buttons and text ran to the edge of the screen and nothing looked right. On an actual Apple Watch, however, interface elements were given a sense of space, thanks to the hardware around the screen.

Designing for TV is the exact opposite. If you push your interface elements up against the edges of the screen, you're likely to lose them to any or all of the typical television safe zones. Smart TV apps must respect these varying soft areas around the outside edge of televisions in order to prevent cutting off text or buttons. It's just like print design.

Considering a video aspect ratio of 16:9, you can expect a resolution of 1920 x 1080px for television in most cases. But only the background images of your smart TV application should be expected to reach the full height and width of that screen. The action safe area is actually 1792 x 1016px, and the title safe area is even smaller at 1720 x 1000px, which is where you should plan on keeping content and interface elements.

Mobile first Watch your favourite TV show and buy a car from your mobile phone? It happens more often than you think

in the previous example. Perhaps you're watching a cookery show on television and want to find the recipe being cooked to save for later. Complementary usage comes into play here, and it's a big part of how we engage with the world. In fact, 77 per cent of TV viewers use more than one device at the same time, and nearly half of those viewers are using a smartphone.

The opportunity here to help people engage with the things they see on television is huge. Watching TV is no longer just a passive behaviour; we're always looking to learn more about the people, places or things we see.

77 per cent of mobile searches happen at home or at work, and 22 per cent of searches done from a smartphone were prompted by something seen on TV. That translates to a lot of mobile engagement, regardless of whether the people themselves are out and about. If your site or app isn't mobile-optimised, you're already missing out.

CHECK YOUR ENVIRONMENT

A person is generally two or three feet away from their computer monitor, and even closer to their smartphone. But for television, the average viewing distance is typically between 12 and 15 feet (often called the '10 foot experience'). This context matters, especially with TV, because doubling the viewing distance essentially halves the apparent size of an object, making 12 point type unreadable. To truly understand a device's limitations, you must experience it in its native environment. For example, designing a smart television app on a computer won't give you the same experience as seeing it for yourself at the right distance, especially when it comes to establishing things like the correct font and image sizes.

We should also be thinking about the physical environment surrounding any device interactions, not just viewing distance. Users sitting at a desk are mentally prepared to work through tasks using devices with the most capabilities (keyboard, mouse, connected devices). But sitting on the couch in front of a television lowers our tolerance for hangups, so simplicity and ease of use are even more important.

Exercise ring The Apple Watch displays fitness information in a way that is understandable at a glance

Moving about Switching between your Apple devices has become even easier with a feature appropriately called Handoff

START WITH MOBILE

By now you've heard of the mobile-first approach to RWD. Starting with mobile forces us to begin with the most constraints, to really think about the content and functionality we need to offer to our users. What will people absolutely need to do, and what's the easiest way for them to do it?

The other advantage of starting with mobile is that most people have a smartphone (and plenty consider it their only computing device). Mobile phones now outnumber people on this planet. So you can count on many of your users to be coming from a mobile device.

MAKE TRANSITIONS SEAMLESS

Considering how often we switch devices, moving from one to another should be less painful than it is. Cloud technologies such as Dropbox and iCloud are addressing this issue, but our software isn't there yet. Apple has decided this is worth looking into, and its Handoff technology in OS X and iOS (*netm.ag/handoff-272*) already allows apps to share data for easy transitions from one Apple device to another.

Nine out of 10 people use multiple devices to accomplish a goal, and almost all of them move between devices in the same day. It's our job to make that transition a smooth one.

THE SCREENS ARE COMING

Beyond the growing number of screen-based internet devices, designers must also consider the devices in our daily lives that don't have screens. This growing Internet of Things is bringing the power of connectivity into our home appliances, cars and even out onto the hiking trail.

> Designing mobile-first forces us to begin with the most constraints. What will people absolutely need to do?

Activity trackers like the Jawbone UP and Fitbit display very little visual information, relying instead on separate mobile apps and the vibration feedback on the device itself. And how do we make the data collected by these smart devices useful? Our software must make sense of it and present it in a way that is easy to digest and beautiful.

The Apple Watch uses an 'exercise ring' to communicate activity data. A quick glance at the watch face and you can see where you are with your calorie, exercise and stand goals for the day. You're welcome to tap on the rings for a more detailed view, but this overview is more than enough in most cases.

TALK AMONGST YOURSELVES

Finally, remember that your users don't use their devices in a silo, so you should avoid designing in one. Without open communication in the workplace, your product cannot succeed in the hands of your users. Make sure design teams throughout the entire company are sharing their experiences, goals and user research, and everyone is on the same page.

Design thinking must be done outside the box, and the screen. Today it's about evolving from app-focused design to an action-based ecosystem of connected devices. The future is now.

PHOTOGRAPHY

JOBY SESSIONS

jobysessions.com

ILLUSTRATION

STEVEN WILSON

Steven is an illustrator who lives and works in Brighton, UK. He creates varied and experimental work which he describes as 'somewhere between pop and psychedelia'

stevenwilsonstudio.com

AUTHOR

BEN CALLAHAN

Ben is the president of Sparkbox and founder of the Build Right workshop series. He writes about the web on leading industry blogs, and speaks around the world

seesparkbox.com

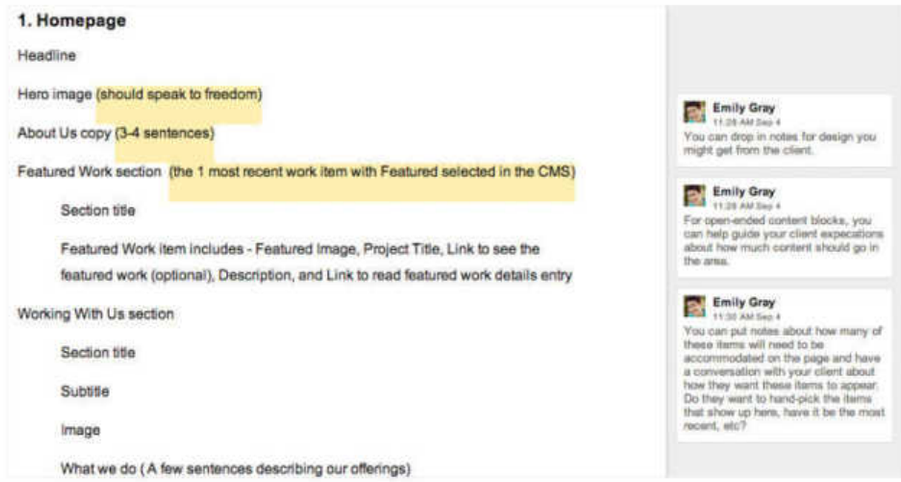
ONSIVE DESIGN

BEN CALLAHAN RUNS THROUGH THE HOTTEST TOOLS YOU NEED TO BUILD SITES THAT WORK ON ANY DEVICE

My name is Ben Callahan, and I'm a tools junkie. If I'm not careful, I'll spend more time playing about with the hottest new tools than doing the work they were built to help me with. Chances are, some of you are the same way, which is why I was excited at the opportunity to experiment with, and ultimately recommend, some of these tools.

These days, every new site I visit is responsive. It feels like this flexible approach to building for a multi-device web has become the norm rather than the exception. To this end, if you consider it your job to build a more future-friendly web (*futurefriendlyweb.com*), then all of these tools will help you accomplish that goal. Some of them are more specific to the kinds of challenges we face in responsive work. Some are less directly related, but still quite relevant.

Finally, these tools are not ordered by priority. Instead, I've tried to select tools that span all the tasks we face, from design to development. So comparing them is difficult and, frankly, not very useful. Right, let's jump in …



Content Priority Guide An example guide from our recent redesign. You can see a working version at netm.ag/redesign-268

1 CONTENT PRIORITY GUIDE

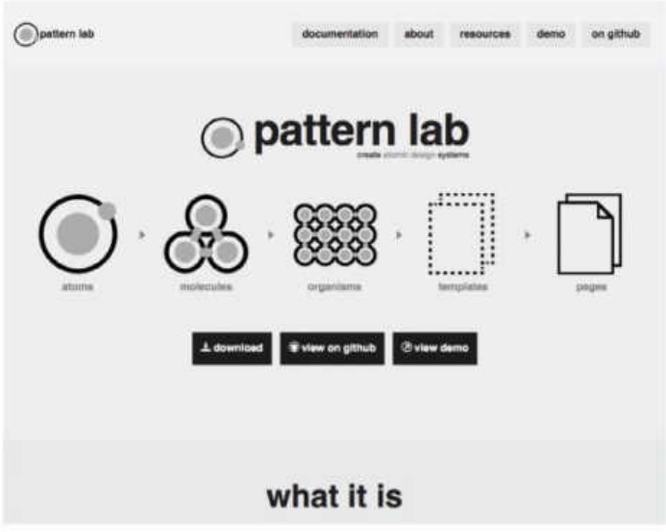
netm.ag/priority-268

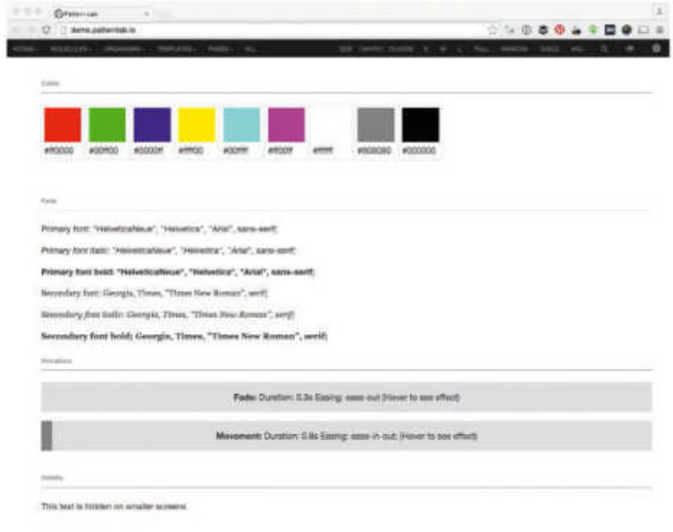
When I first started making websites, I was happily oblivious to the importance of content in building for the web. I quickly learned that I was missing a critical part of the process – I was like a sculptor with no clay. If you find yourself struggling to incorporate content strategy techniques into your process, this simple tool is the perfect way to introduce some content-first thinking into your workflow.

The Content Priority Guide was created by Emily Gray as a kind of hybrid tool to use in situations where it's not possible to get all the content in, but you need to begin some form of UX design.

"As a project manager, I needed a solid place to start UX design. But as a content strategist, I didn't have all the content, only the ideas behind the content. I developed the guide to fill in those gaps," explains Gray. The concept mixes ideas from content modelling (*netm.ag/modelling-267*) with simple wireframing (*netm.ag/wireframe-268*).

In order to use this guide, you need to walk through each unique page in your system, identifying the applicable content types and breaking them down to their smallest parts. Additionally, Gray creates these as a Google Doc and shares them with her clients. This encourages a more collaborative approach when it comes to establishing content types, as well as starting the process of prioritisation. In responsive work, understanding the





Sketch The Sketch interface was built from the ground up for web design, not photo editing or print design

hierarchy of content and functionality, independent of viewport width, is critical. This tool helps you establish that early on in the process.

Content priority guides help you identify reusable content types, which leads to a solid understanding of how to structure your CMS. And, if you're using modern frontend development techniques (see tool 2), it also gives you a framework for how to modularise your markup.

2 PATTERN LAB

patternlab.io

Dave Olsen and Brad Frost have developed a beautiful tool for facilitating pattern-driven design. Pattern Lab is based on the idea that you should break your design down to the smallest parts ('atoms') and then use those to construct larger, reusable components ('molecules' and 'organisms'). These three core building blocks can then be used to create templates, and injected with real content to create pages.

At its core, Pattern Lab is a static site generator. In practice, it's much more. The structure that this way of building provides results in a much more modular system – one that's easier to integrate with any content management system.

The structure that Pattern Lab provides results in a much more modular system – and one that's easier to integrate with any content management system

3 SKETCH

bohemiancoding.com/sketch

When it comes to design, a lot of folks are talking about Sketch. Where tools like Photoshop and InDesign have their roots in design for other mediums, Sketch was built from the ground up with the web in mind.

Allowing for multiple pages per file and multiple artboards per page makes the task of organising your design significantly simpler. Features such

PERFORMANCE BUDGETS

In response to the criticism of many slow responsive sites, there's been a lot of discussion around ways to integrate performance thinking earlier in the process. If you consider that around 60 per cent of the weight of web pages is in images, you have to concede that designers can have a tremendous impact on the performance of a site. For this reason, I'm very excited to see performance-minded designers like Yesenia Perez-Cruz and Katie Kovalcin speaking (*netm.ag/perez-cruz-268*) and writing (*netm.ag/kovalcin-268*) about performance budgets.

This idea is actually quite simple – it's really about establishing parameters within which to work. At Sparkbox, we've started using the WebPageTest API (*netm.ag/test-268*) with our Gulp-based frontend build process. This way, our designers and developers are notified immediately when they implement a feature that pushes us over the established performance budget.

In addition, Tim Kadlec has built a Grunt task for enforcing a performance budget. There is an implementation of this inside the Pattern Lab interface to notify you there when your pages get too heavy. The tooling here is growing rapidly. Find something that works for you and your team and stick to it!

If you want to dig deeper into performance budgets, check out Tim Kadlec's fantastic write-up on the metrics we use to measure performance at *netm.ag/kadlek-268*.

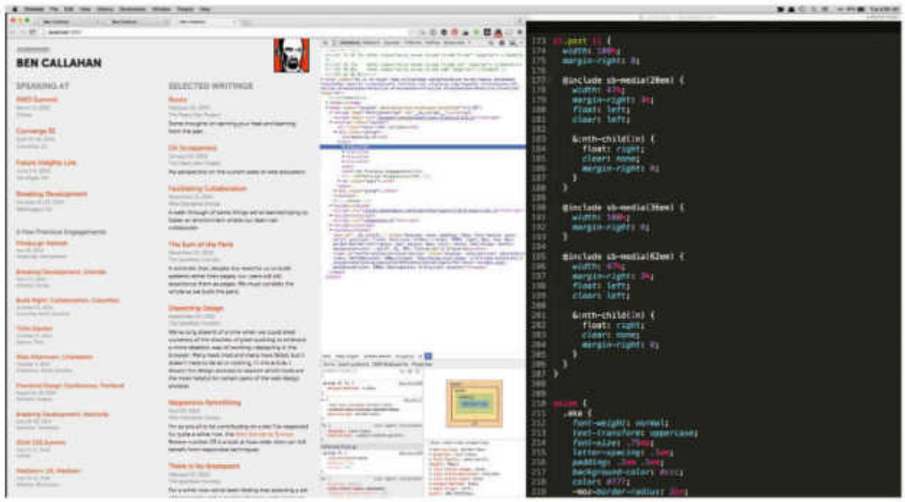
EXPLORING EMPATHY

In preparation for writing this article, I reached out to the Twitterverse to ask what tools people found indispensable. While I received many great answers – some of which are included in this list – there was one response from my friend Dan Rose (@dblizzy) that really stuck with me. He replied with just one word: empathy.

This word is about as loaded and overused as any other I've heard in this business. Mostly, I think we use it in reference to our users. We talk about putting ourselves in the shoes of our users so that we make better design and experience decisions. This is all absolutely fantastic. However, I've been thinking a lot lately about the empathy that we have for our teammates. As our process shifts away from linear handoffs and towards a more iterative approach, we need to deeply consider how the decisions we make will impact the others on the project.

I've also sensed a 'fracturing' of the roles involved in making the web. To be honest, I'm worried that this is leading us to create fractured experiences. The cure for this is empathy – it's having some level of understanding of the other roles needed to do the work we do.

If these ideas resonate with you, I've shared more about them in a recent post on The Pastry Box: *netm.ag/roots-268*.



Sass A screenshot of my desktop while editing my personal site. You can see the use of the sb-media mixin (replacing the @media keyword) and style nesting in the code to the right

as mirroring allow you to easily view your work on connected iOS devices, taking the element of guesswork out of designing for small screens.

Sketch has tools for both vector and raster editing, but seems to be set up in favour of vectors. Couple this with its simple and powerful exporting features and you're ready to support the variety of high-pixel-density screens out there now, and (fingers crossed) whatever tomorrow brings.

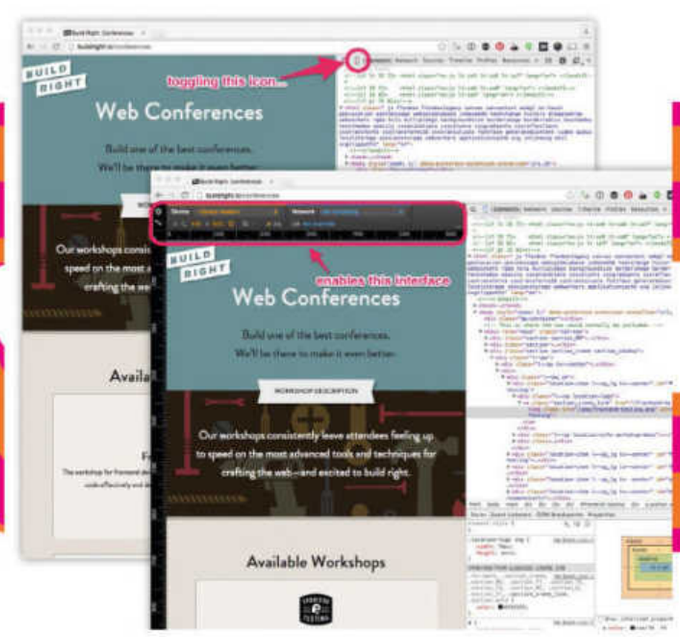
4 SASS

sass-lang.com

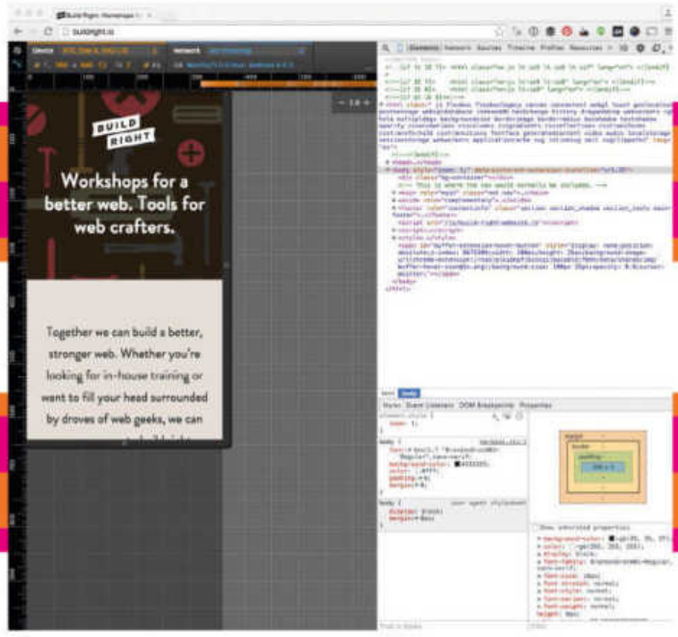
I remember being sceptical of CSS preprocessing when it first started to show up. I'm a bit old school – I've been writing CSS for a long time now – so the idea that my CSS was going to be generated by some 'preprocessing event' made the hairs on the back of my neck stand up. It also didn't help that I was fairly disappointed with the generated CSS from the first few iterations of Less and Sass.

But, like most of you, I've since seen the light. Sass has given us things we've always dreamed of: variables, nesting and maths to name a few. But it's also opened up whole new ways to work with features like mixins, media query bubbling and the ability to create partial and aggregate files (find out more about all these at *netm.ag/reference-268*). I can honestly say that Sass has completely

Network simulator Selecting from the network throttling drop-down will allow you to simulate various network speeds



Device mode Chrome's device mode reveals an amazing set of features



Screen emulator This enables you to emulate the screen of a specific device

changed the way that I work with style on the web. I create more modular and maintainable code (*netm.ag/modular-268*) and much of that is possible (or at least, much easier) because of Sass.

5 POSTCSS

github.com/postcss/postcss

If you think CSS preprocessing is cool, you should check out what's being done with PostCSS. The idea is simple: it's a JavaScript-based tool for 'transforming' your CSS. The most popular PostCSS plugin seems to be Autoprefixer (*netm.ag/autoprefixer-268*), which parses your CSS and adds vendor prefixes to the necessary rules by checking Can I Use (*caniuse.com*) for support. This makes a large portion of frameworks such as Compass (*compass-style.org*) unnecessary (although Compass does do some other pretty amazing things, like creating your sprites for you).

What I love about the concept of PostCSS is that we can polyfill any of the coming CSS features based on their not-yet-implemented specifications. This is incredibly exciting and you can do it today using cssnext (*cssnext.github.io*) – another PostCSS plugin.

If you think CSS preprocessing is cool, check out what's being done with PostCSS. The idea is simple: it's a JavaScript-based tool for 'transforming' your CSS

6 CHROME DEVTOOLS

netm.ag/devtools-268

I will never forget the first time I used FireBug to debug my CSS. The ability to modify, in real time, the HTML and CSS on my page completely blew my mind. It also revolutionised how I worked on the web. Since those days, I've made the shift to Chrome and the Chrome DevTools.

Just as most modern browsers are constantly being developed, so are the developer tools that ship with them. Recently, Chrome has added a few features which I've found very useful in responsive web design. Here, I want to highlight three specific features which I think are worth looking at.

These are most easily accessed by toggling 'device mode' in the Chrome DevTools. You get there by inspecting the page and clicking the little 'phone' icon in the top-left of the inspector, next to the search icon.

NETWORK SIMULATOR

Poor performance has been a huge criticism of responsive web design over the past few years. Some of the reasons for this are outlined in a post by Guy Podjarny (*netm.ag/perf-268*). While I find myself more aligned with Tim Kadlec, who suggests the real fault is in the implementation, not the technique (*netm.ag/kadek-268*), our implementations obviously need some work (see 'Performance Budgets' boxout). Part of the conversation needs to be around tooling that helps bring performance into the conversation earlier. Chrome's network simulator does just this.

With the network simulator, you can – you guessed it – simulate various network speeds. The idea itself isn't new; we've been able to do this for a while now with other tools. However, having this ability right in Chrome DevTools brings it immediately to the forefront of the development process – where it should be.

SCREEN EMULATOR

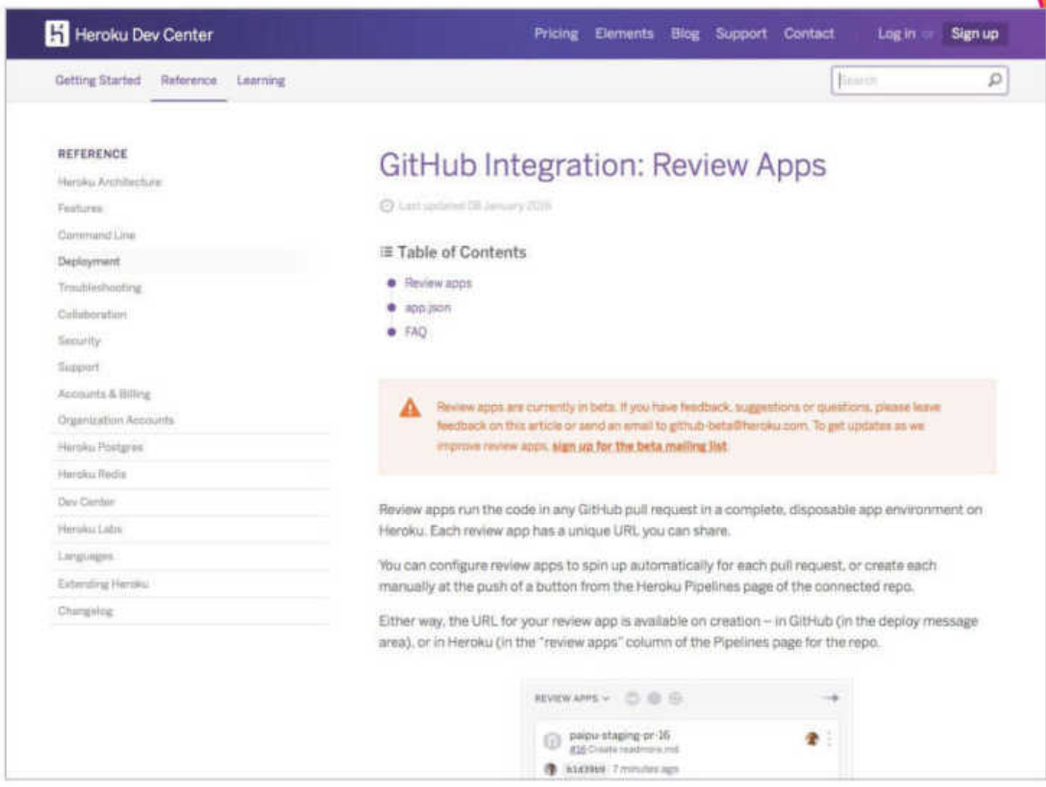
The screen emulator does more than just change the size of the viewport to match

the size of a specific device – it also specifies the user agent string and pixel ratio. There are a handful of very useful presets, and you can also create your own and save them for later use. For more information on how Chrome's device mode works, check the Chrome developer site (*netm.ag/emulator-268*).

MEDIA QUERY INSPECTOR

Much of our work in responsive web design involves the serving of style adjustments to specific subsets of media types and features. You and I call those media queries, and the media query inspector in Chrome's DevTools makes it very easy to dig into those queries.

You can bring this feature on-screen by clicking the 'media query' icon. Assuming the site you're inspecting has media queries, you'll see a series of coloured lines across the top of the inspector. Blue represents `max-width` queries and orange represents `min-width` queries, while green represents query widths within a range. Clicking these coloured bars resizes the mock-viewport and shows you the width at which those styles apply. You can also right-click the coloured bars to reveal where the queries they represent live in the source. All quite useful!



Review Apps Heroku's GitHub integration simplifies the pull request process, helping you get your code out quicker

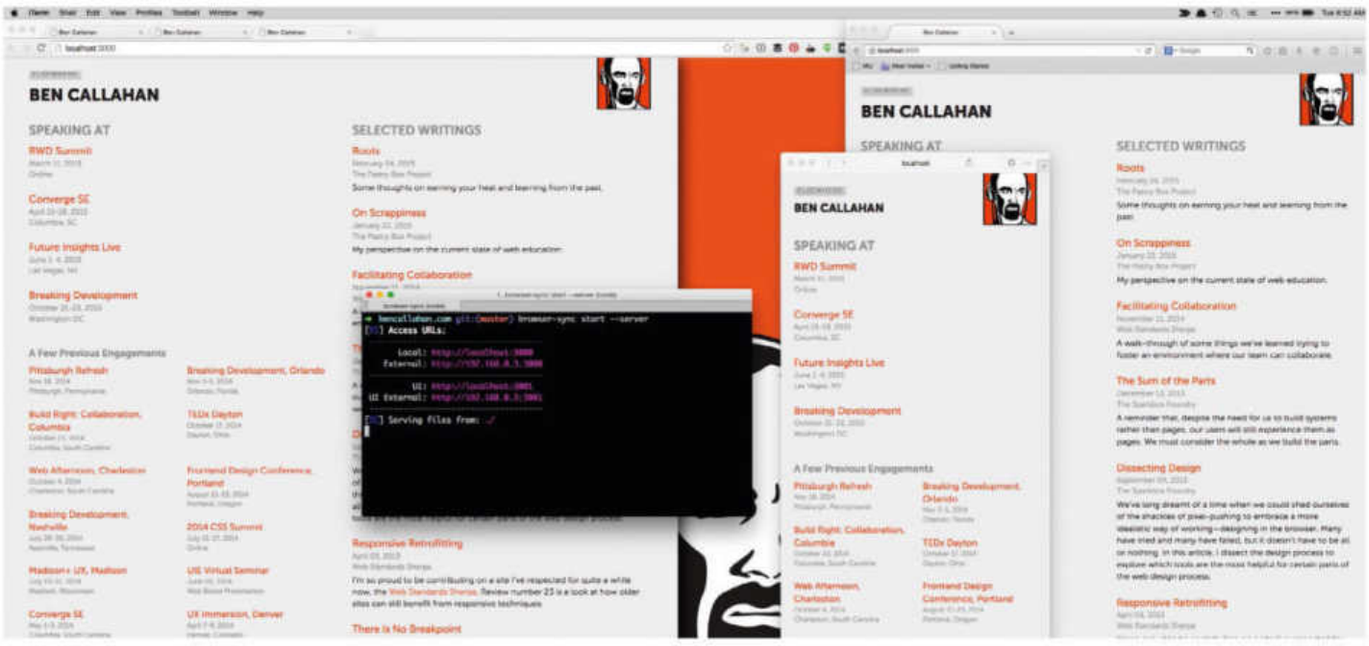
7 HEROKU REVIEW APPS

netm.ag/heroku-278

Review Apps is a GitHub integration from Heroku. The reason I'm including it in this list is the ease with which it helps you get your working code onto devices to test and into users' hands.

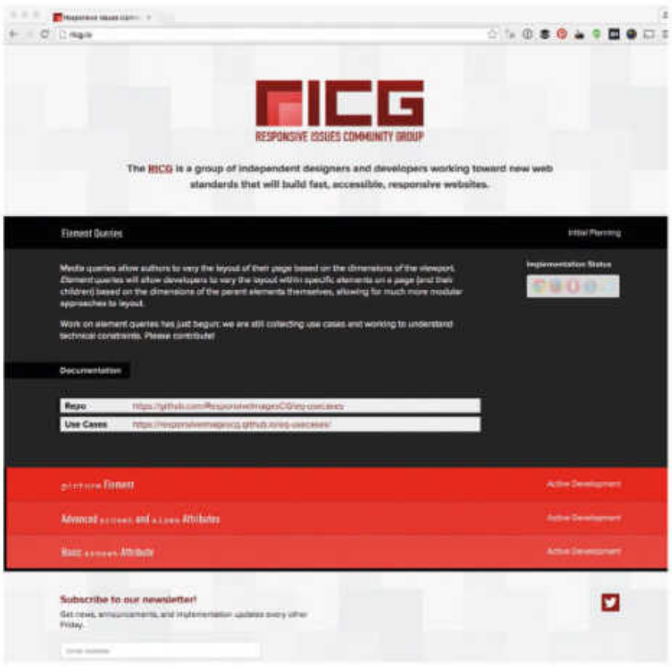
When properly configured, Review Apps allows you to have a publicly available version of your site or app running for every pull request (PR) submitted to GitHub. That means anyone at your organisation can test individual PRs before they are merged, and without having to spin up a local development environment. Additionally, if you choose to merge that PR, you can follow it up with an automatic deployment to another environment (like production).

Note that this is currently in beta, but we've been using it during development on many projects, with great success!



Browsersync A screenshot of my desktop while testing my personal site using Browsersync. Chrome, Safari, and Firefox are all syncing scroll position, clicks and more

Skitch Skitch by Evernote is a fantastic screen capture and annotation tool that found its way into my workflow in a matter of minutes



RICG The Responsive Issues Community Group (formerly, the Responsive Images Community Group) has forged a new path in bringing our community together to create change

8 BROWSERSYNC

browsersync.io

One big challenge these days is how we accommodate the growing population of devices on the market. Even if you only focus on the top six or eight devices, testing and debugging can feel like a serious hindrance to your productivity.

Enter Browsersync; a free, open source tool for testing and debugging sites across all the devices in your testing lab. Not only does it sync the URL of any browser, it also syncs scroll position, clicks, form inputs, toggles and submits, and refreshes or injects CSS as you make changes. It comes with a slick UI (available by default at *localhost:3001*) and includes the open source project weinre (*netm.ag/weinre-268*) which enables you to remote debug any of the devices connected to your localhost.

Browsersync has an official Grunt plugin (*netm.ag/grunt-268*) and works seamlessly with Gulp. It's also got an API for those looking to develop other integrations. When you're dealing with today's web, Browsersync is seriously useful. *[For more on BrowserSync, check out our tutorial on page 198].*

9 SKITCH

evernote.com/skitch

Skitch is a native app for Mac, iPad and iPhone (sorry, all you non-Mac users), which makes grabbing and annotating screenshots amazingly simple. It's built by the folks at Evernote, but you can use it even if you're not an Evernote user.

I've included it here because of how easily it allows you to provide feedback on a prototype. Plus, you can drag straight from Skitch into GitHub issues to attach your annotated screen capture. We use Skitch relentlessly when we're testing. Providing helpful, detailed feedback has never been easier.

Not only does Browsersync sync the URL of any browser, it also syncs scroll position, clicks, form inputs, toggles and submits

10 RICG

ricg.io

My final tool is not actually a tool at all. The folks that participate in this group are largely responsible for the new `<picture>` element, along with the new `srcset`, and `sizes` syntax. What has me really excited, is that the next project they intend to tackle is a solution for element queries.

Element queries will enable us to selectively apply styles based on the width of the containing element, not just the width of the viewport or device. Combine this with the ideas of pattern-driven design and new specs like Web Components (*webcomponents.org*) and the future is looking pretty bright!

WRAPPING IT UP

It's a great time to be working on the web. The landscape is challenging, but we're finding good solutions. Our tooling is evolving as fast as the problems we're trying to solve.

In the end, though, tools alone are not enough. Remember that most of the reasons projects fail are not for technical reasons, but because of the people involved. Now, go make the web!

SUBSCRIBE TO NET

GET THE NO.1 MAGAZINE FOR WEB DESIGNERS AND DEVELOPERS DELIVERED TO YOUR DOOR, YOUR DEVICE, OR BOTH

PRINT EDITION ONLY

Take out a print subscription to **net** and get your copy before it hits the shops. Each issue is packed with the latest web trends, technologies and techniques

FROM
£55

SAVE UP TO
29%

DIGITAL EDITION ONLY

Take out a digital subscription to **net** for on-the-go access to our interactive edition, with streaming screencasts, extra images and more

FROM
£45

SAVE UP TO
31%

Terms & conditions: Prices and savings quoted are compared to buying full priced print and digital issues. You will receive 13 issues in a year. If you are dissatisfied in any way you can write to us at Future Publishing Ltd, 3 Queensbridge, The Lakes, Northampton, NN4 7BF, UK to cancel your subscription at any time and we will refund you for all unmailed issues. Prices correct at point of print and subject to change. For full terms and conditions please visit: *myfavm.ag/magterms*. Offer ends 31 Dec 2016

PRINT & DIGITAL EDITION

Enjoy a combined print and digital subscription, and take advantage of print as well as exploring the digital experience on the go

GREAT REASONS TO SUBSCRIBE

- Print edition delivered to your door
- 13 issues in a one-year subscription
- iPad and iPhone edition download
- Android edition download
- Money-back guarantee

FROM
£66

SAVE UP TO
54%

myfavouritemagazines.co.uk/NETMAG16

generate

The conference for web designers

NEW YORK 22 APRIL 2016

Learn cutting edge CSS, JavaScript, user experience and web performance techniques, and much more!

CAMERON MOLL
DESIGNER AND AUTHOR
www.cameronmoll.com

DAN MALL
CREATIVE DIRECTOR
www.superfriend.ly

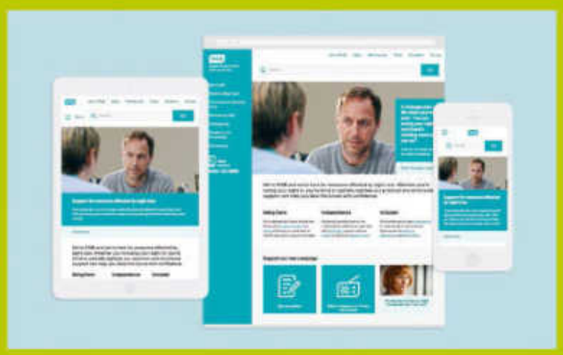
SARAH DRASNER
SENIOR UX ENGINEER
www.trulia.com

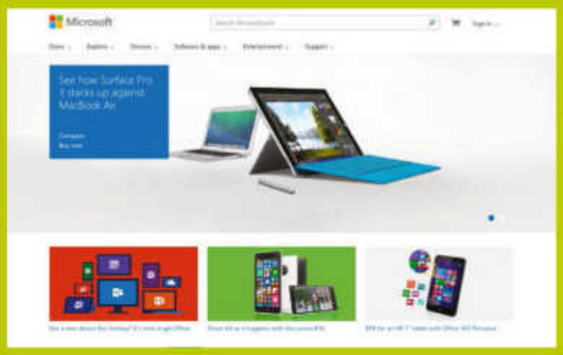
UNA KRAVETS
DEVELOPER
www.ibm.com

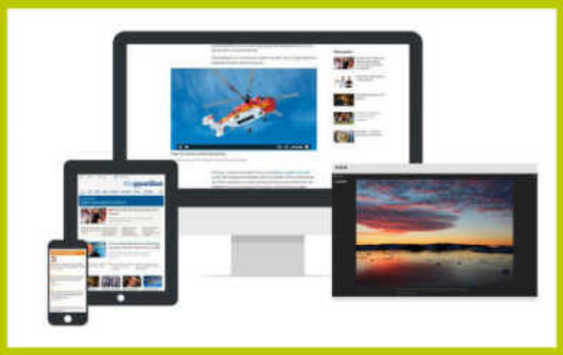
TICKETS ON SALE NOW

www.generateconf.com

SHOWCASE

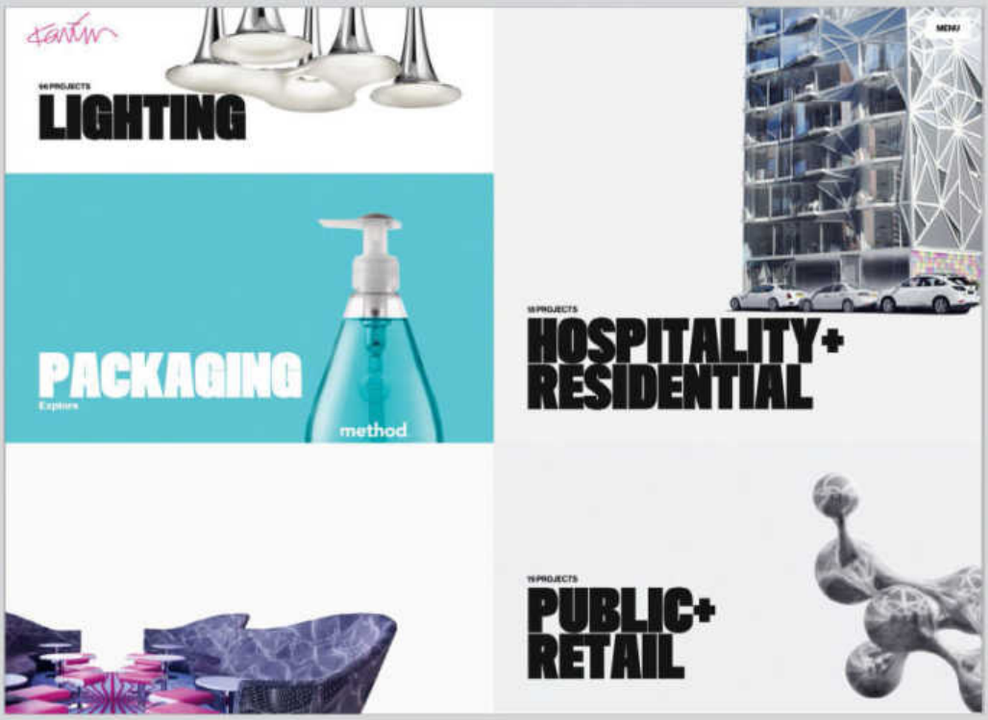






GALLERY

Sensational design and superb development



★ ANIMATION, JAVASCRIPT

KARIMRASHID.COM

Anton & Irene *antonandirene.com*

Designer Karim Rashid is as prolific as he is celebrated. With over 3,000 designs in production and work in 20 permanent collections worldwide, it is perhaps no surprise that his own website had taken a back seat for the past 10 years. Until he commissioned Anton & Irene, that is.

With its trademark embrace of engaging interactions, emotive typography and truly device-agnostic layouts, the duo has delivered a site that invites vigorous exploration. Which was, of course, the point: "Our main goal was to showcase the plethora of products and projects in a visually enticing way," say the pair on their blog.

This plethora of products also presented a challenge. How does one deliver 5,000-plus images of varying shapes and sizes in a responsive environment? The solution: a re-imagining of conventional layouts and a lo-fi approach to structuring to give a precise, flexible grid.

Perhaps the most impressive thing about Rashid's new site is the seamless transitions between viewports. "We always design all screens simultaneously ... The moment we have an idea for a component or a layout we try it on all screens and see if it makes sense across the board."

Words by Espen Brunborg

"OK, OK, admit it. Karim Rashid's new site rocks"
GARY PIKOVSKY (@DESIGNTIMES)

ESPEN BRUNBORG

Espen is a creative director and co-founder of Primate, a small web agency that creates innovative online products in Edinburgh
w: *primate.co.uk*
t: @ebrunborg

UNA KRAVETS

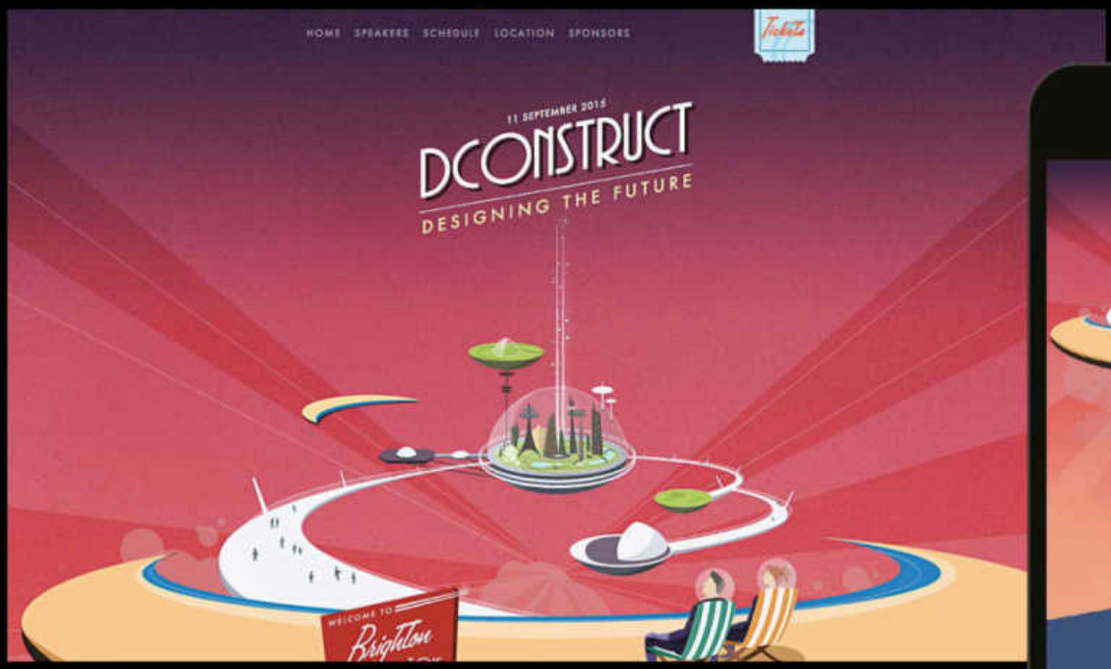
Una is a frontend developer, architecting design systems and building software prototypes at IBM Design in Austin
w: *unakravets.com*
t: @Una

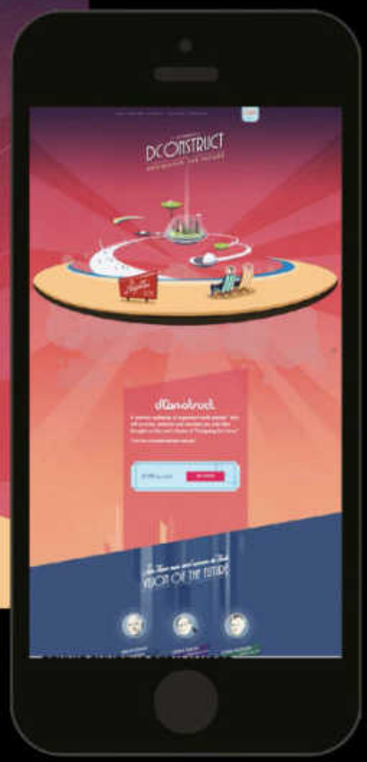
LYZA DANGER GARDNER

Lyza is co-founder and CTO of Portland-based Cloud Four, where she works with a phenomenal team to take the web into the future
w: *cloudfour.com*
t: @lyzadanger

JUSTIN AVERY

Justin is a digital consultant by day, and by night creates the RWD Newsletter, hosts a RWD podcast and runs a RWD knowledge hub
w: *responsivedesign.is*
t: @justinavery





CSS TRANSFORMS, CANVAS ANIMATION, SVG

2015.DCONSTRUCT.ORG

Clearleft *clearleft.com*

The site for 2015's dConstruct conference in Brighton is a happy place to explore: "part 1920s Metropolis and part 1950s Jetsons," according to Andy Budd at Clearleft. It's sparkly. It animates gently. It's awash with jaunty angles. The colours phase hot, then cool, then hot again as you tour through the content.

Yet it's less than one kilobyte on a primed cache. It scrolls like butter. Web fonts keep the look spunky and stylised, but they're loaded asynchronously for better performance. Under the covers the site's markup has the serenity of a Rothko painting – not surprising, coming from Clearleft, where semantics and HTML craft are forefront.

"The hardest part was getting the diagonals working," says Budd. "Once we'd nailed that, the rest was just tweaking the designs and animations."

It's so solid and comfortable in various viewports that, from the receiving end, it doesn't feel pinned to a particular platform, but is at home on a whole lot of them.

Words by Lyza Danger Gardner

> "The site is responsive, fast, and sports an aesthetic that's at once striking *and* welcoming. And those subtle CSS skew transforms are just stunning"
> ETHAN MARCOTTE (@BEEP)

★ JQUERY, ANGULARJS, GOOGLE MAPS

CODECONF.COM

GitHub *GitHub.com*

The site for CodeConf really goes above and beyond the standard conference website. I was lucky enough to also attend the event as a speaker, and I must admit I was extremely impressed with the thorough branding (even the food was very in-theme). The conference was held in Nashville, Tennessee, and everything about this design pays homage to this location.

The website itself is nicely responsive and has a warm, cohesive colour palette. The whimsical illustrations give the site character and create a playful country-rock aesthetic that continues throughout the page (and even into the event itself).

No details are spared, as even the menu's decorative horizontal rules (only seen on smaller screen sizes) flow with the country-rock aesthetic. The site implements Google Maps for location features, and is built with jQuery and AngularJS.

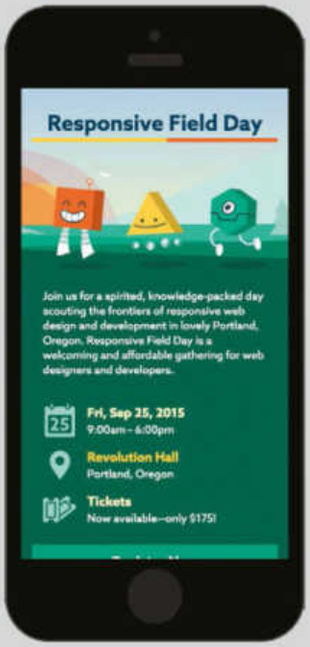
Everything is illustrated: all of the venues, the 'set list' of speakers, the calls to action, and breaks between sections. There is also a fun cast of characters that can be found dotted around the site: vector cacti, unicorns, dragons, octocats, and cowboys and girls playing music and posing playfully around the page.

Words by Una Kravets

SVG, GREENSOCK, SUIT CSS

RESPONSIVEFIELDDAY.COM

Cloud Four *cloudfour.com*



This event site was created by the very clever folks at Cloud Four. Coming from this amazing team, you would expect a pretty awesome implementation. They did not disappoint.

"We wanted it to feel focused but also friendly and inclusive, with some of Portland's inherent strangeness intact," says lead designer Tyler Sticka. "The quirky, toy-like robots frolicking along the Northwest landscape represent individuality, fun and future-friendliness."

The team has used a grid framework that I haven't come across before: SUIT CSS. This provides a great foundation without all the extras you sometimes get with Bootstrap and similar frameworks.

The CSS is beautifully crafted with *lots* of comments to indicate why each element was used. You could argue the team was being lazy in not removing it – however, I like to look at it as a fantastic learning example that would help anyone getting into writing CSS see why and where particular rules are included.

Words by Justin Avery

GREENSOCK, ANIMATIONS, RWD, ILLUSTRATION

CROPTRUST.ORG/2014

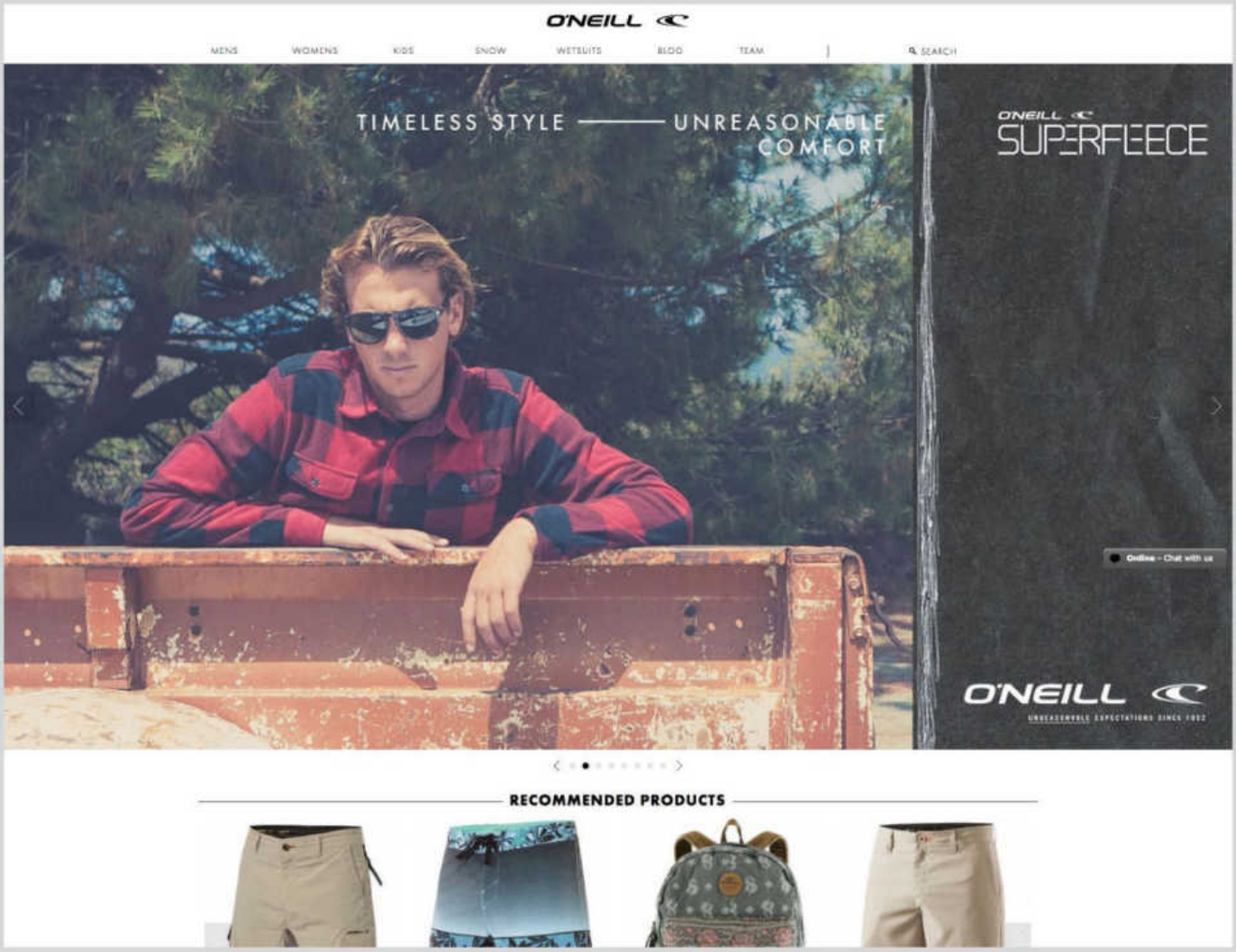
Epic *epic.net*

The Crop Trust is an international organisation working to safeguard crop diversity. Although that sounds like literally watching grass grow, the site is playful and interactive while also being informative. The team at Epic – the agency behind the site – has used some great fullscreen photography to bring the story to life.

GSAP has been used for animations, including the Canvas-based triangles that tie back to the logo; and the transition between the report homepage and supporting content. These animations are subtle and add to the experience. They are left out of the mobile experience, which is all about the content.

One of the biggest challenges for Epic was the amount of content it was required to deal with: the website would act as the Crop Trust's primary source of information. Its solution was a three-tiered structure. "The chapters and key figures were organised as panels to provide a quick and very visual overview of the report's content," Benoit Rondeux from Epic explains. "We then piled up the chapter pages so users can read each topic one after another, like a book. Atop of that, we grouped all the highlights of the year (news, events and so on) in a compact and visual timeline, accessible from the panel view."

Words by Justin Avery



*ECOMMERCE, RESPONSIVE IMAGES, GRUNTICON

US.ONEILL.COM

Electric Pulp *electricpulp.com*

O'Neill has really nailed the commerce website market. It approached its redesign with a simple aim: "We wanted the new site to have both purpose and soul on the smallest screen through to the largest." I think they nailed the brief several times over.

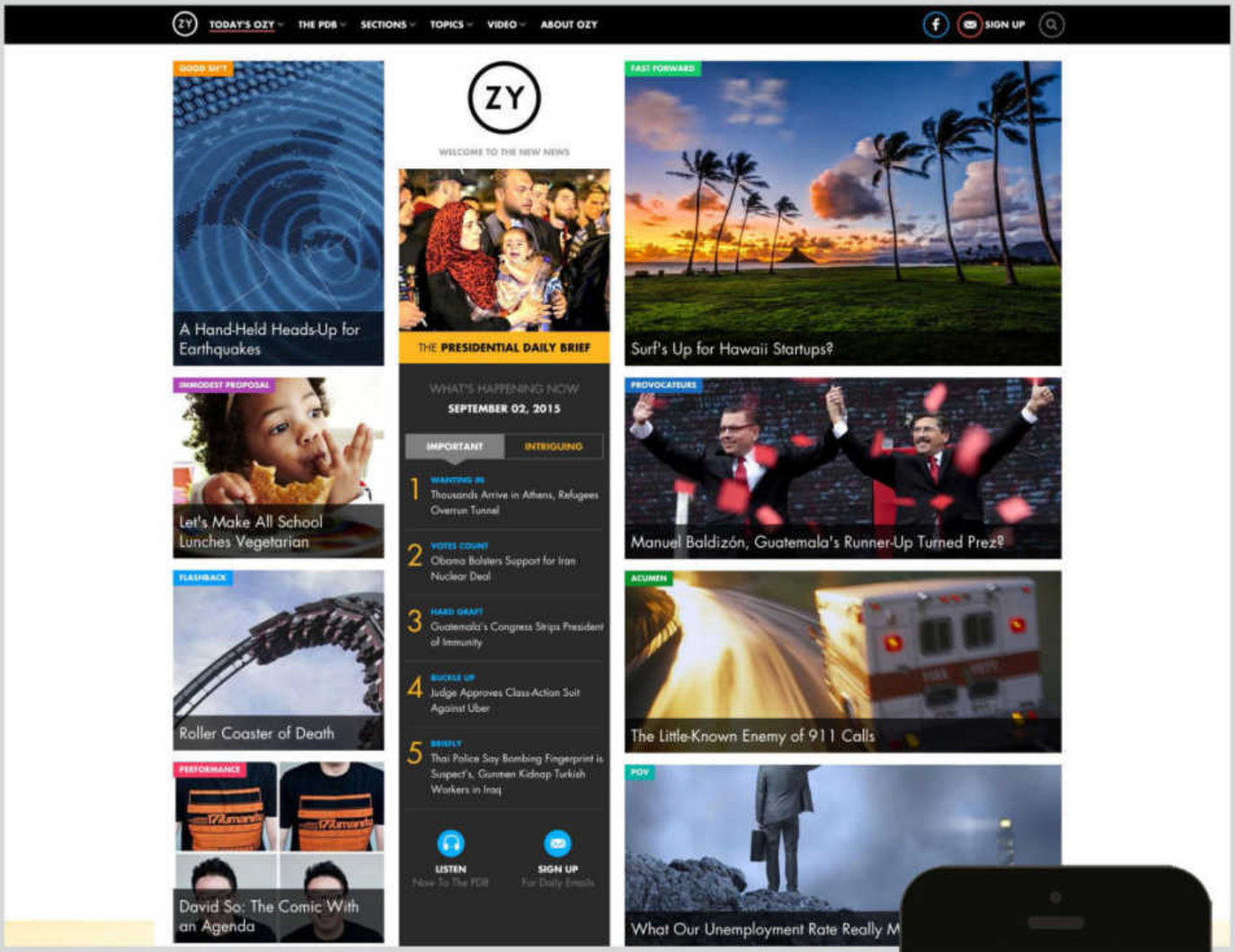
Built by the team at Electric Pulp, the new site is a star example of the potential impact of RWD: after the responsive version of O'Neill launched, it saw an increase in conversions of 65 per cent on the iPhone and over 400 per cent on Android, and a revenue increase of 101 per cent and 591 per cent respectively. Put that in your responsive pipe and smoke it!

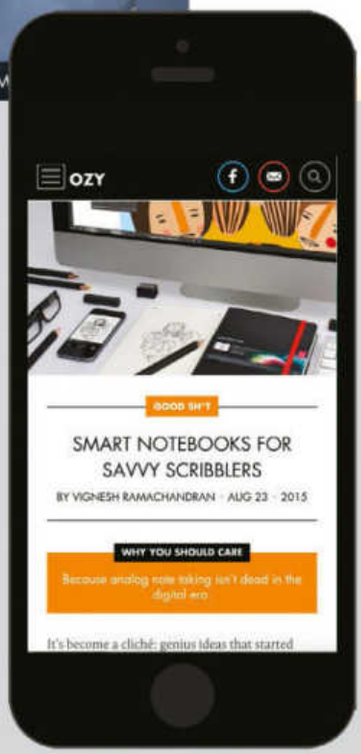
The website is stunning in its visual simplicity, providing enough white space to allow each product to shine. Although it is very image-heavy, the first paint is a passable three seconds.

The site uses an implementation of the `picture` element and `srcset` to target the `x` descriptor only. SVG is key when it comes to simple, crisp iconography on any responsive site. Here, Grunticon is employed as a fallback to ensure everyone gets a similar experience.

Words by Justin Avery

> "Dang. O'Neill's redesigned (and responsive!) storefront is looking SHARP"
> ETHAN MARCOTTE (@RWD)





LESS, RWD, LAZY LOAD

OZY.COM

OZY *ozy.com*

OZY is an example of how to execute a great responsive experience for a news site. The mobile-first approach means the site loads the mobile version of the images to begin with, and then if the user's device benefits from a larger image, this is loaded using `data-src` attributes.

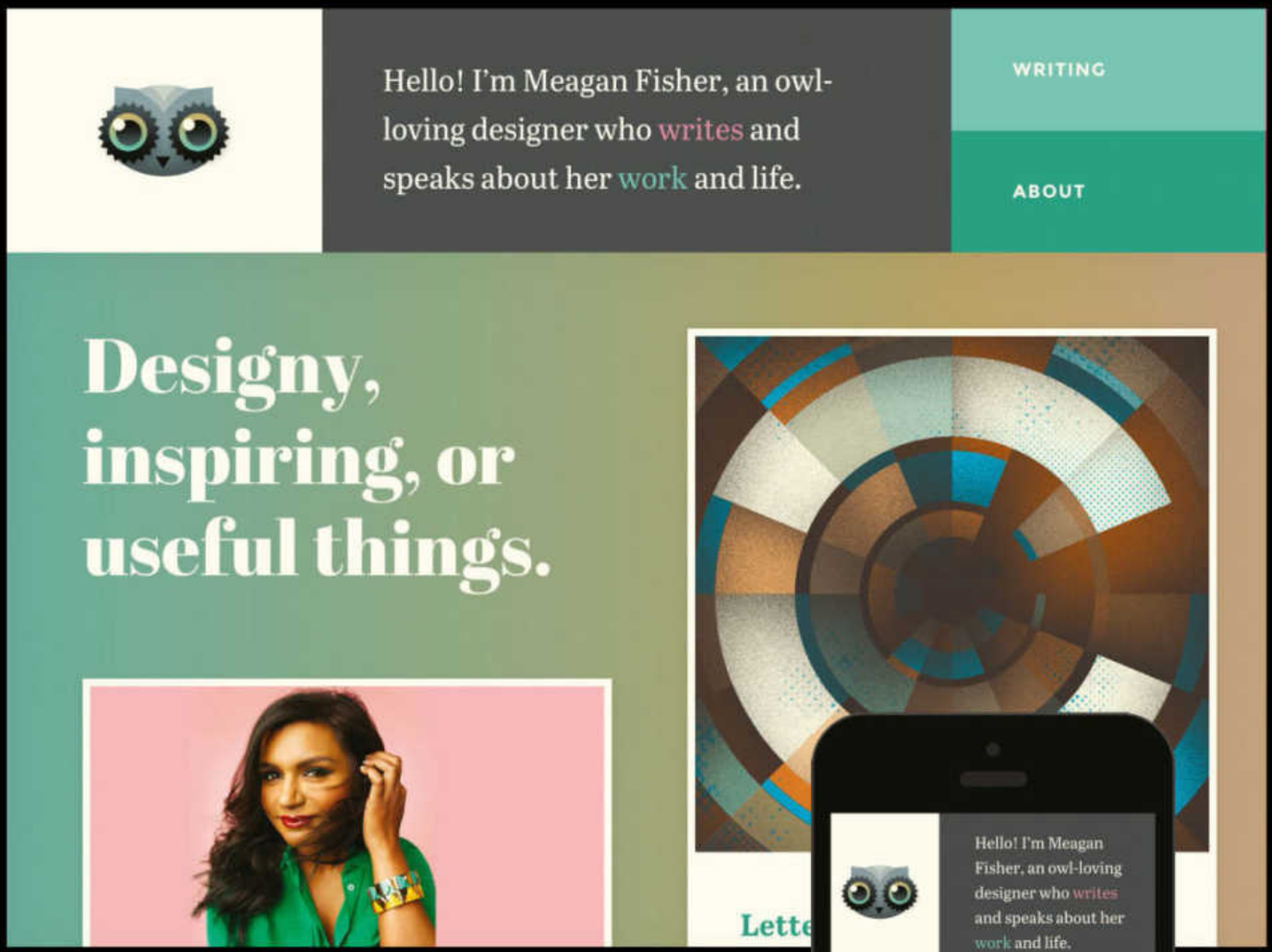
The reading experience is great: there are three well-constructed grids that span across seven different breakpoints, to ensure the content is always the focus as you move between devices.

One of my favourite parts is the 'Most Read' section. Here, all the HTML content is ready to be read, but images are lazy loaded to save bandwidth and ensure the content is immediately available.

OZY also implements an infinite scroll to enhance the reading experience. Moving from article to article on the mobile is seamless, but when the scrolling would have a negative effect on performance (like with iOS7 or older IE versions), this is disabled and recommended articles remain as standard links.

The simple touches of colour are just one of the subtle design decisions that help make this one of my favourite sources of content.

Words by Justin Avery



*TYPOGRAPHY, RWD

OWLTASTIC.COM

Meagan Fisher *owltastic.com*

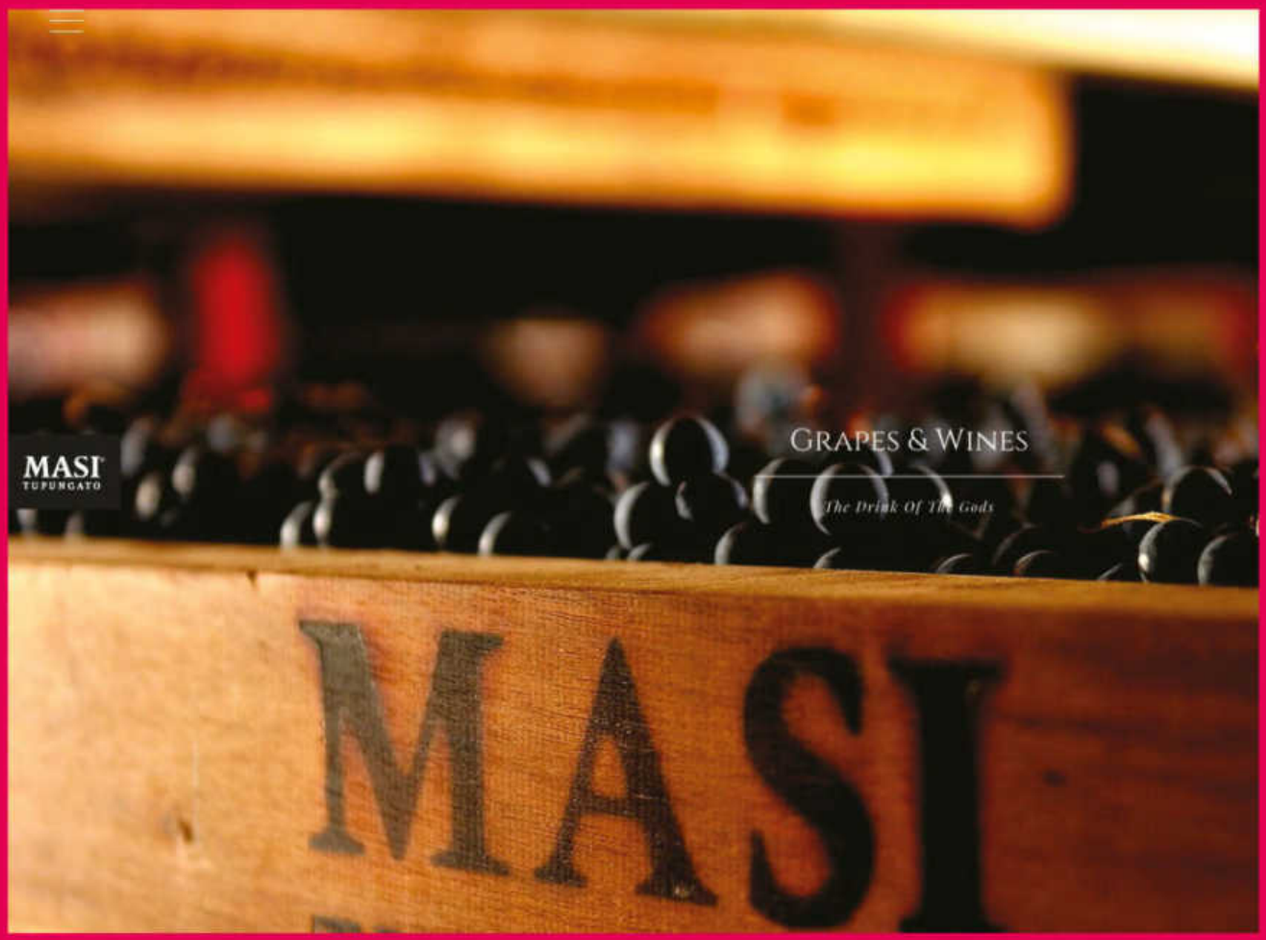
This is a wonderfully fun portfolio site and blog from the talented designer Meagan Fisher. Fisher has strayed away from the popular approach of hamburgerising the navigation, instead opting to stack it … and it works really well.

The design is a big departure from Fisher's previous portfolio site. "I wanted something that felt more like me," she explains. "With the last design I had a very stripped down site, and this left it feeling more like a template than my online home."

The breakpoints give each piece of content on the page enough space to breathe across a variety of devices, and design touches such as drop shadows and gradients make the site a delight to browse. Fisher has selected the perfect complimentary fonts and paid attention to measure and line length, so reading is a pleasure.

"My goal was to let the site be a playground for the colour, imagery and layout experimentation," she smiles. "I hope it will continue to evolve as my style does."

Words by Justin Avery



CSS3 ANIMATIONS, RWD, VELOCITY.JS

MASITUPUNGATO.COM

AQuest *aquest.it*

This wonderful website from international digital creative agency AQuest for Masi Tupungato, a winemaking project based in Italy, almost lets the imagery speak for itself.

Unusually, a loading screen is used for each of the pages as the crisp fullscreen images load up. Typically this would be a big no-no – users want the content as soon as possible. However, here it actually improves the user's experience by ensuring images are fully loaded before any content is unveiled. The design leaves users feeling like they've been to the winery and picked the grapes themselves.

The site can be on the heavy side on some pages, which could be improved by introducing some lazy loading techniques. However, despite its weight, the site is well-built, with the start render in under one second and return visits loading within the second mark too. The framework is based on *unsemantic.com*, which is a successor to the 960 Grid System.

When viewing the site on desktop and larger viewports, users are able to see and interact with each of the wines separately. They can take advantage of the larger screen size to display all of the wine characteristics and details side-by-side. In contrast, on the mobile site the details and description slide in and can be slid away again smoothly.

Words by Justin Avery

"Masi Tupungato has created a beautiful example of seamless design and slick performance. A thoroughly enjoyable experience"
CHRIS BURNELL (@IAMCHRISBURNELL)

SVG ICONS, PICTURE ELEMENT, VIDEO ELEMENTS

ROYALALBERTHALL.COM

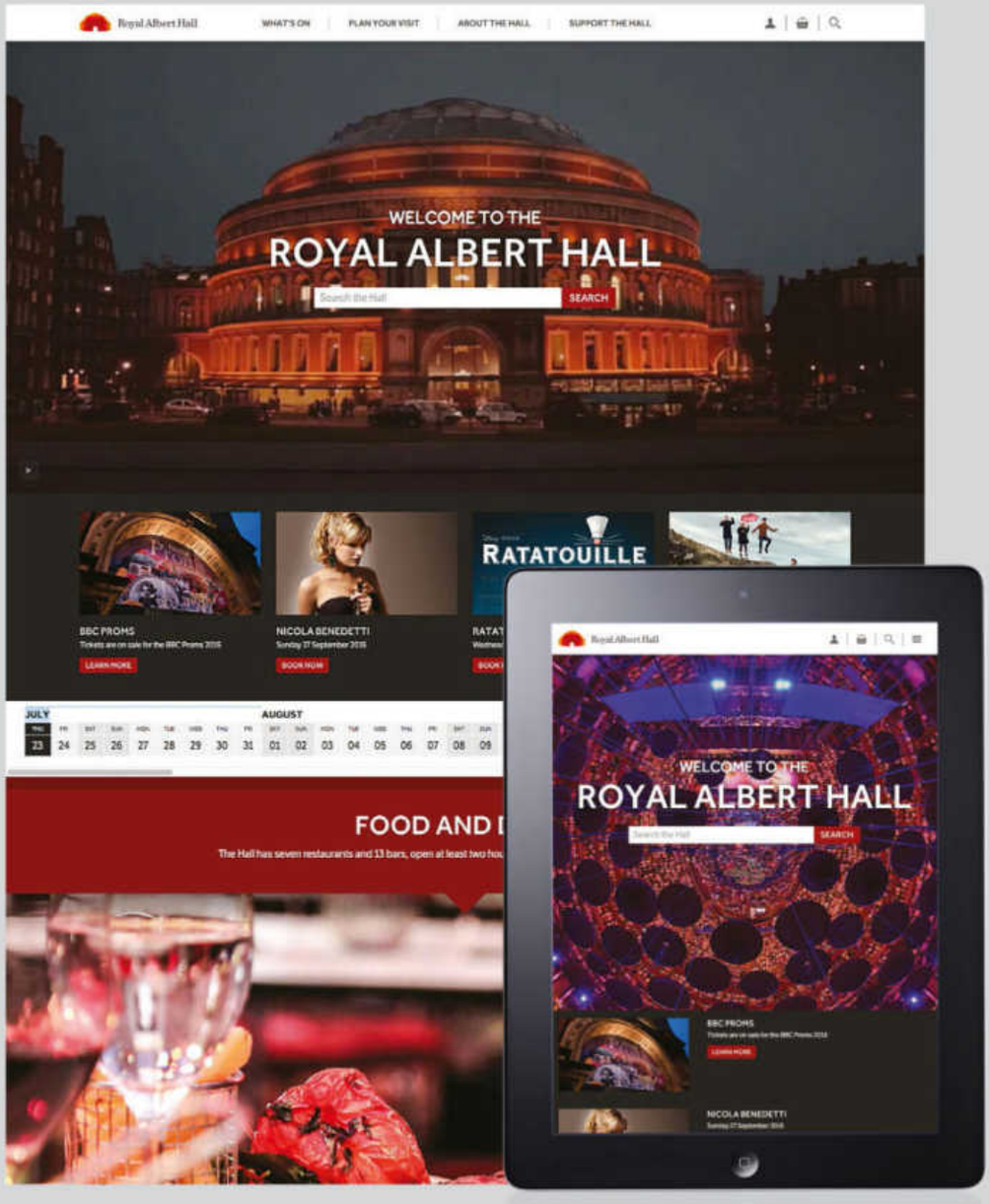
Made Media *mademedia.co.uk*

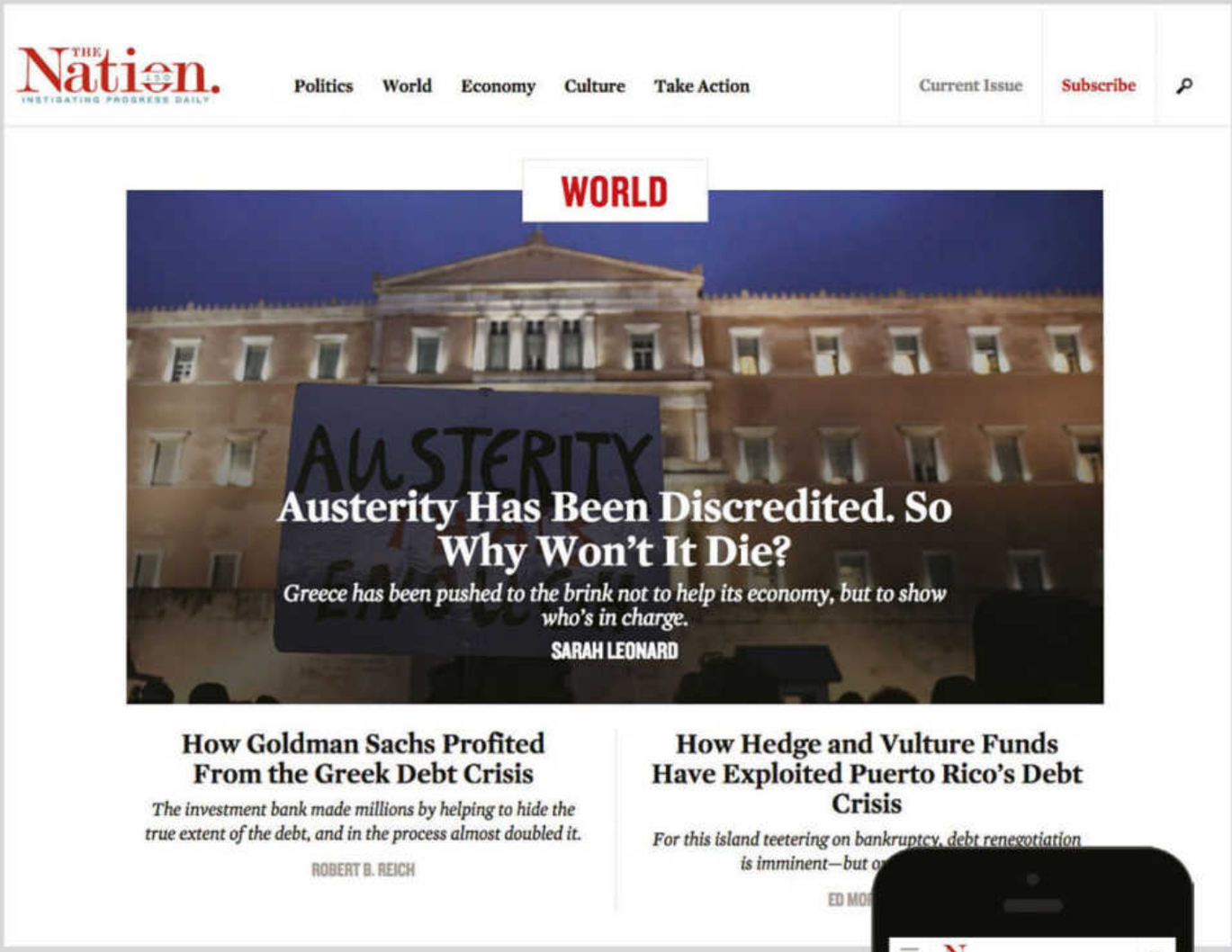
The new Royal Albert Hall site is responsive across the gamut, but especially so with media, making use of responsive images (using the picture element) as well as progressively enhanced video and SVG icons.

UK agency Made Media spearheaded the project, which features lush images of the hall, combined with crisp typography. Visitors with wider viewports on capable browsers get a full-width video front and centre, in which London traffic and gathering concert-goers subtly flow around the iconic building.

The design is thoughtful even across the sometimes-neglected middle viewport widths, and all layouts share the same bold black-grey-red colour scheme, with snappy contrast.

Words by Lyza Danger Gardner





Showcase

WEB FONTS, WORDPRESS

THENATION.COM

Blue State Digital *bluestatedigital.com*

The website for weekly US magazine *The Nation* has been given a big responsive revamp, and it has certainly been noticed. It's one of a growing number of established publications that has faced up to the challenge of modularised content and cross-device chaos.

Blue State Digital – the digital strategy agency known for running Obama's 2008 and 2012 digital action plans – was responsible for pulling off the hefty feat of adapting the deep visual traditions of a 150-year-old magazine for the modern web.

Hoefler & Co.'s Mercury Text is in strong showing here – a font specifically crafted for newspapers, with an adapted screen variant. Design nuances echo the brand's words-on-paper roots, with pull quotes and hairline rules providing a cadence reminiscent of print, without feeling contrived or fragile.

The challenges here are numerous: content from many sources, advertising elements, media. But we're seeing a trend in which, increasingly, larger publications are facing these trials boldly, head-on.

Words by Lyza Danger Gardner

FOCUS ON

FULLY RESPONSIVE

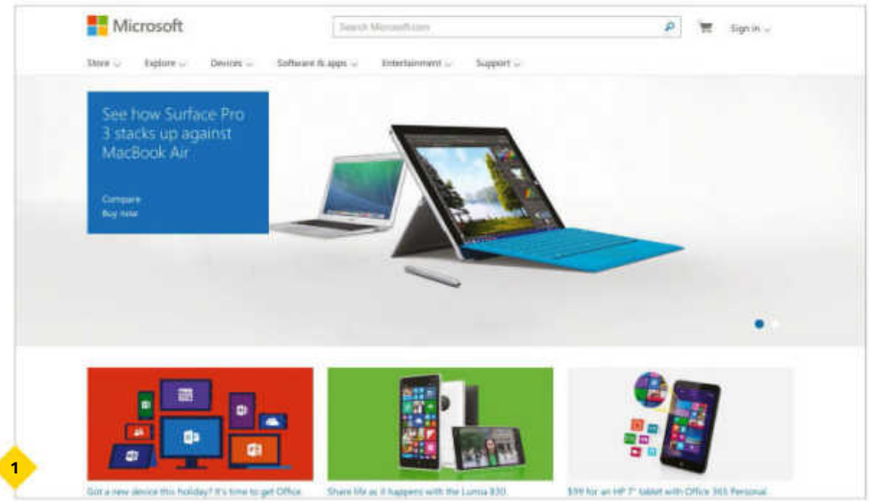
Gene Crawford explains why, in order to make your sites truly responsive, you need to look beyond the breakpoints

As the main editor of *UnmatchedStyle.com*, I review a lot of websites. And while most of them are 'responsive', there is still a very large majority that are not what could be called 'fully responsive'.

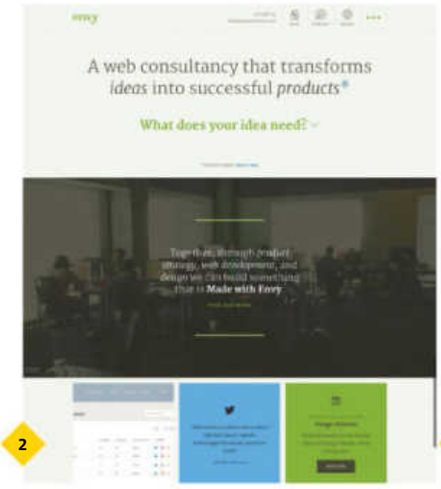
It's worth stating here what makes up a responsive website. A website is responsive if it utilises all three of the following: flexible images, media queries and a fluid, proportion-based grid. When done properly, there's true magic here. When you apply this responsive approach to building, alongside thinking mobile-first, you essentially future-proof your website.

It is not enough to grab a few specific media queries that target popular devices and design for those. There are so many different devices – mobile and otherwise – out there in the world, that you simply can't take them all into account. You have to design for the in-between stuff. The only way to do that is to think mobile-first when you are applying these responsive techniques to your projects.

Herein lies the beauty that we, as web designers, bring to the table. What we do is more than simply making something pretty or visually slick. We get to take things deeper and work with how we communicate to the audience with the content, the visuals and the interactions, across many different mediums. It is a lot to keep on top of, but it is what separates us, as web designers, from other disciplines of design.



1



2

3

(**1**) The Microsoft (*microsoft.com*) website shifts beautifully between known, device-specific screen widths and everything else that 'could be' out there. (**2**) Web design and development firm Envy (*madewithenvy.com*) scales its website's elements well no matter what screen width you may have. It also utilises some known device breakpoints for major navigation design changes. (**3**) With its responsive design approach, the website for web design and development agency Erskine (*erskinedesign.com*), is a great example of working between breakpoints.

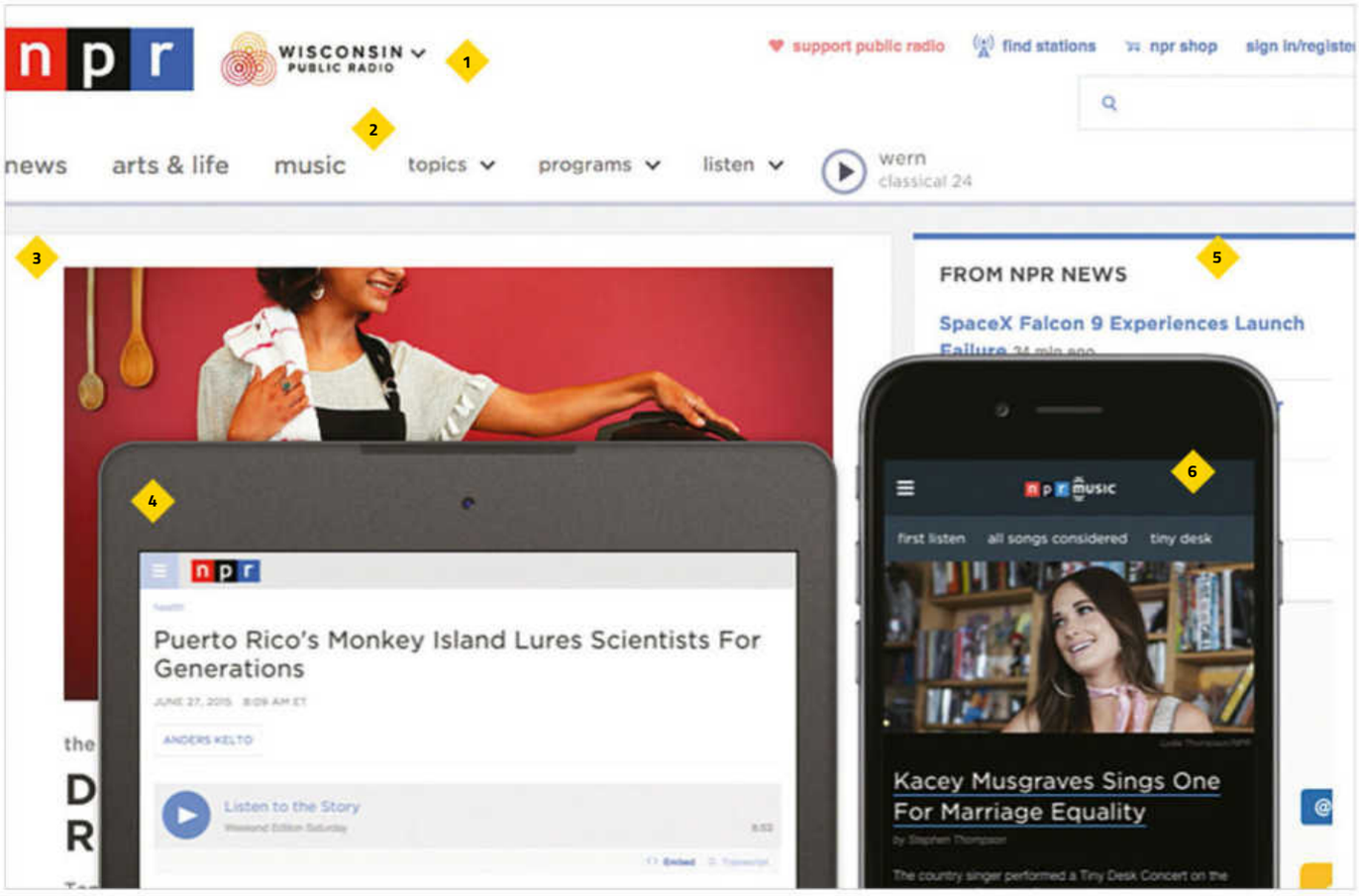
PROFILE

Gene's mission is to work tirelessly to provide inspiration and insight for developers. His recent projects include *unmatchedstyle.com*

HOW WE BUILT

NPR

National Public Radio used an agile approach and comprehensive user testing on its three-year journey to a fast, responsive new site



BRIEF

NPR wanted a coherent, curated and wide-ranging content experience on the web – whatever the screen size or device. An in-house team worked to build connections between NPR and its audience, and encourage users to listen, read and watch.

CLOSE UP

(**1**) The site localises users to their closest public radio station. This feature helped increase average session duration, reflecting the connection users have with their station. (**2**) Navigation was designed to offer quick access to content sections. These three links also succinctly communicate the breadth of NPR's content: news, arts & life and music.

(**3**) Many sections present stories on virtual 'cards', each offering the option to listen, read, watch or interact. (**4**) We simplified and tuned the design of the individual stories to create an optimal reading experience, and to make it easier to press 'play'. (**5**) User testing confirmed the importance of a dense, scannable list of news stories to give the audience a quick overview of what's happening in the world. (**6**) The NPR Music section deserved a distinctive visual treatment, but maintains a familial connection to the overall site design.

PATRICK COOPER

Patrick oversees product strategy and agile development of the NPR site
w: *patrickcooper.com*
t: @btrpkc

SCOTT STROUD

Scott is the lead interaction designer and user researcher for the NPR site
w: *scottstroud.com*
t: @sstroud

TODD WELSTEIN

Todd is the primary user interface developer for the site's responsive design
w: *toddwelstein.com*
t: @toddwelstein

For the responsive redesign of the National Public Radio website (*npr.org*), the team embraced a patient approach, allowing the revamped site to gradually emerge over three years of agile development. Each portion of the site presented a different problem, and the solution to each issue revealed new information for the next cycle. Using a variety of prototyping and user testing methods, the team worked to build a site that would become a frequent guest in its users' everyday routines.

What was the aim of the redesign?
SS: In the earliest phases of the project, the designers were passionate about eliminating excess. Over the years, NPR had given site editors the power to tell stories expansively on desktop browsers. Unfortunately, reconstructing those complex narratives on smaller screens became difficult. More flexibility in layout didn't ultimately help users.
TW: We also wanted to retire our old mobile *m.* site to eliminate this path of software maintenance and architecture.
PC: The product team's goal was to deepen our users' relationships to NPR and local stations, whether they be committed listeners on the radio or casual visitors accessing the site via a social media link.

Were there any sites that influenced the direction you took?
SS: We were inspired by different news organisations at different times: the early responsive redesign of the Boston Globe, the clarity of the Guardian and the media-rich experience of the BBC ecosystem. We also evaluated newer sites focused on visual storytelling, like Medium, Vox and NOWNESS.

What sort of user research and prototyping did you do?
PC: NPR has a long history of user research, so we gathered past artefacts and reports about our audience. The team studied our latest statistics on device usage to remind ourselves how, and how often, our pages are viewed on mobile phones and tablets.
SS: We prototyped our site designs using various tools (paper, static images, HTML/CSS, InVision and After Effects to name a few). These prototypes were valuable for a number of reasons. First, they served as a design tool to help us express detailed interactions and animations. Second, they enabled us to present realistic models to users during usability tests. Third, they were useful in communicating our designs to developers, other team members and various stakeholders.

In combination with agile, we followed lean UX design methods. User research and prototyping were critical parts of our practice – even if we had to make some assumptions to stay lean and keep things moving.

What were the main technical challenges along the way?
TW: Users leave when a site's pages load slowly. We worked hard to optimise the

TIMELINE

The key stages in the three-year project

LATE 2011

NPR runs early experiments in cross-platform consistency, including an adaptive redesign of the live music concerts pages

JUL–OCT 2012

The team makes a case for responsive and launches the most critical element of the system first: the story pages

FEB–APR 2013

NPR builds and launches mobile breakpoint of a responsive homepage, with a stream of hand-selected story summaries

AUGUST 2013

The team releases the new desktop and tablet homepage designs, including a new premium sponsorship unit

DEC 2013–MAY 2014

The product owner prioritises and schedules the responsive conversion of the remaining site sections. The team launches topic and programme pages

OCT 2014

The new NPR Music homepage launches – a result of a collaboration with design partner Frank Chimero

APR 2015

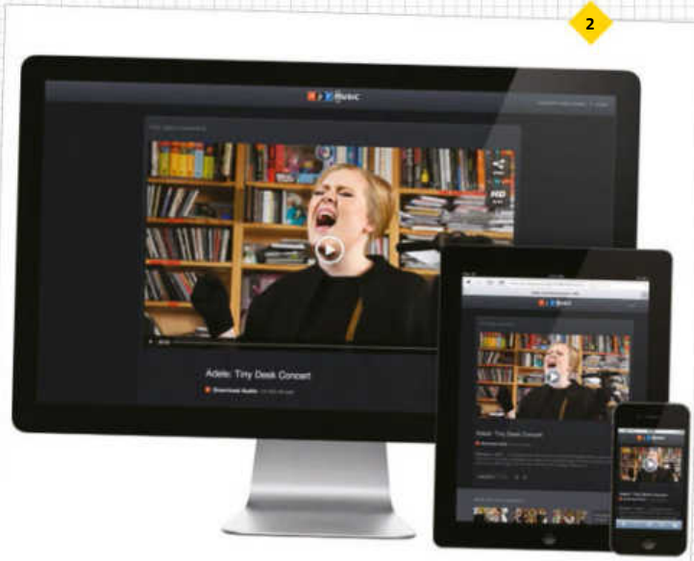
The team launches an embeddable audio player to encourage the distribution of NPR content on other sites and on social media

JUN–LATE 2015

NPR tackles the most complicated part of the project: a persistent audio experience to replace an aging Flash pop-up player

1

2

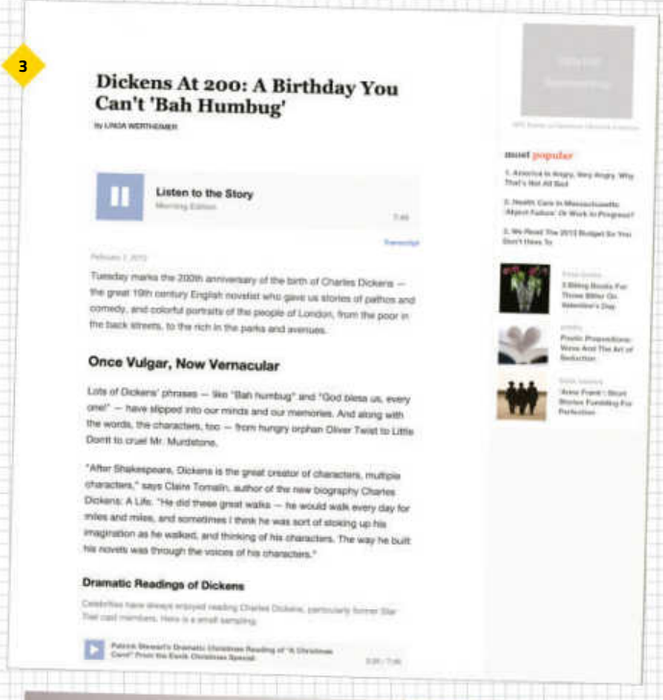


EVOLUTION

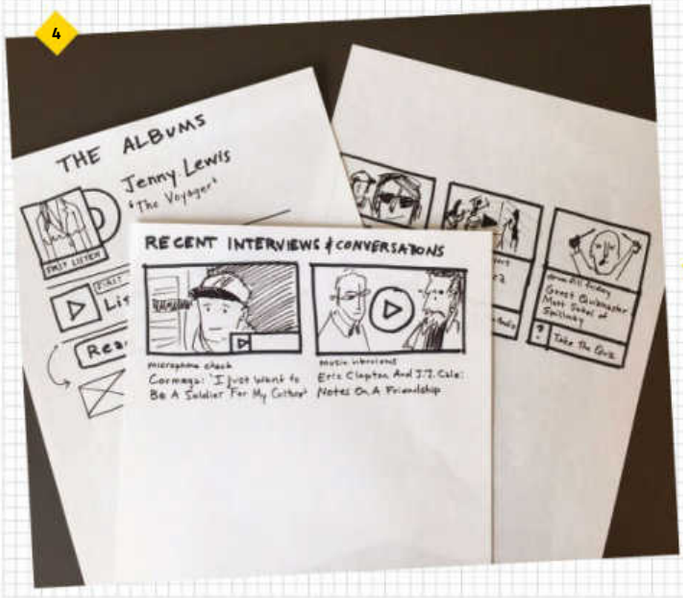
A look at the research, user testing and prototypes that led to the final design

(1) The team reviewed past NPR audience research, including personas. **(2)** Early experiments established a focused, 'lights-down' experience for music concerts. **(3)** The first story page designs focused on prominent play buttons to encourage engagement with audio assets. **(4)** Sketching sessions aimed to inspire team members to communicate ideas about how to solve problems. **(5)** Understanding visual patterns helped improve cross-platform consistency.

3



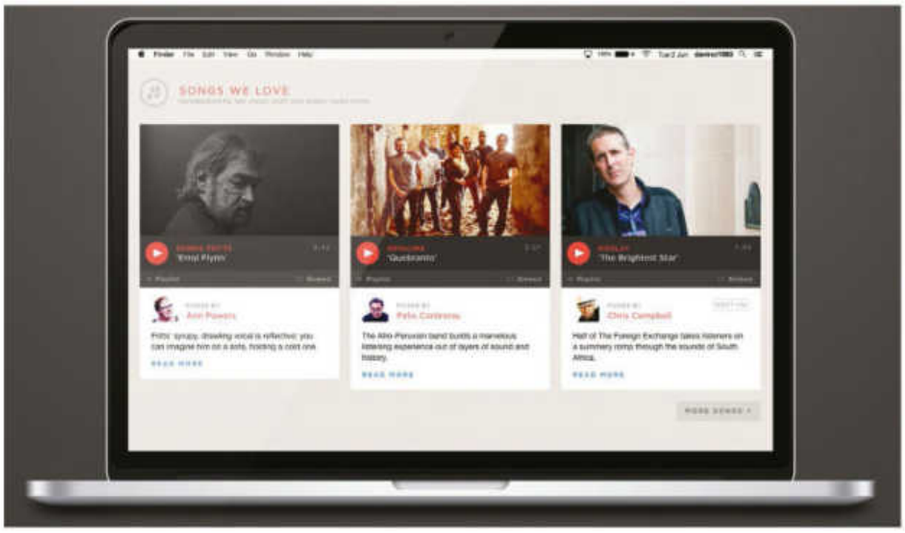
4



5

performance of our site, and as a result we are one of the faster loading news sites on the web.

TW: One challenge was to prioritise reductions in technical debt. With a project extending over several years, the complexity of code increases, along with the inventory of things to improve or simplify. For example, we were able to start with a clean slate for CSS. However, sometimes the HTML couldn't be altered, because it was being shared across responsive and pre-responsive layouts. This meant writing responsive styles against less-than-ideal markup. Now that the site is mostly converted, we have the opportunity to clean up some of the HTML.



Songs we love This section simplifies the delivery of music recommendations by collecting and displaying the best songs from many NPR voices and local station DJs

What was your approach to social media – was there room for innovation?

SS: Though we've included social media links throughout the site for both stories and Follow/Like opportunities, we don't necessarily see users clicking those icons ravenously. We know users want control when it comes to sharing: they often copy the link and paste it into their favourite social media tool or use native browser buttons on mobile devices.

PC: We appreciate how much of our traffic originates from social media, so our design process considers the common experience of landing on a story page first, often in an in-app browser on a mobile device.

Tell us a bit about your appoach to user testing ...

SS: NPR believes we should listen to and observe our users as often as possible, so we've made user testing a standard part of our team's Agile planning. Recruiting users is often difficult, so we've found an interesting method for finding willing participants. People were coming to our building for tours every weekday, so we decided to invite them to participate in user tests before or after their tour. Visitors from near and far bring family members who may not know much about NPR, so we're able to screen for a variety of user types.

We conduct several styles of user testing. Early in each phase of the project, we meet with users to learn more about how to solve the problem. In the middle, our testing is often formative design research – the aim is to stabilise the designs, not to prove whether or not they are successful. Toward the end of each major product development period, we conduct summative testing with more refined prototypes or staged code, to gauge if it has been a success.

How did/do you plan to measure the success of the site?

PC: We measure common statistics like page views, session duration, repeat visits and audio/video plays. However, even if a user comes to our site and digests a considerable and impactful amount of insight or entertainment in a short amount of time with few page views, that's still a success. Just like terrestrial radio, we want to become a frequent guest in our users' daily routines – for whatever amount of time they have to spend with us.

SS: Like many news sites, we depend on the support of our sponsors, and we measure impressions with an ad server. We're always seeking new ways to present sponsored content without compromising our editorial mission or affecting the user's experience.

What feedback have you had?

SS: There was enthusiasm for the simplicity of the new site, but the homepage redesign surprised some users. We made a choice to curate the homepage as a river of story cards, to increase the likelihood that a user will see and hear important items, rather than randomly darting around the page.

Are there any other exciting challenges in the future?

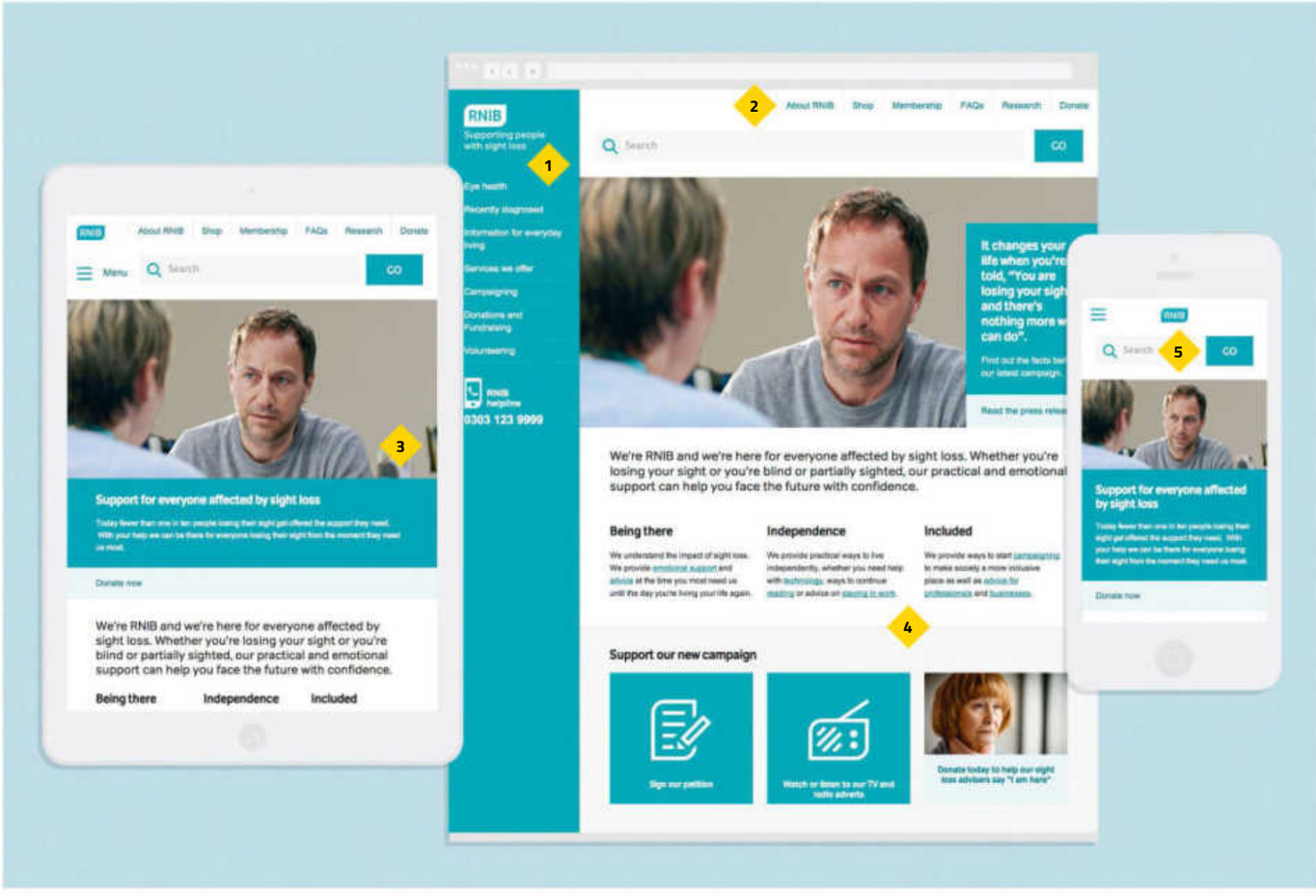
PC: A core feature of our site is the audio experience. However, the redesign of this element is the final major project we're tackling, due to the complexity of the player architecture, persistence, responsiveness and sponsorship. Once completed, when a user hits play they will be able to navigate to other pages on the site without the audio ending and without launching a separate browser window. The player will also support a queue for the uninterrupted listening of multiple stories as users discover and add what they find.

SS: We are currently refining the last stages of the design. Beyond this major release, we're also imagining how a registered user may continue listening to a story across many platforms and devices – as you would expect from Netflix or Amazon TV.

HOW WE BUILT

RNIB

The team at **Precedent** discusses the responsive, accessible site they created for the UK's leading sight loss charity



BRIEF

The Royal National Institute for the Blind (RNIB) is the UK's leading charity for people with sight loss. Digital communications agency Precedent won a tender to design and build its new accessible and responsive website in Drupal.

CLOSE UP

(**1**) Left-hand primary navigation is off-canvas on mobile and flexible enough to enable the user to scale up and down, tab through and read using screen readers. (**2**) Secondary navigation helps keep users better informed right from the start of their journey through the site. (**3**) A large aspirational area on the home page is dedicated to people in real environments, giving a more personal feel to the site. (**4**) Based on a modular system, the new information architecture and design is easy to use and accessible to elevated standards. It enables the charity to effectively deliver many more of its services online. (**5**) The search is more prominent in order to help users get to relevant content quicky and easily. This aims to drive traffic through the site search and will help keep the RNIB informed on the information that visitors to the site are accessing most frequently.

ED RICHARDS

A creative lead at Precedent, Ed creates intuitive designs from complex information architectures.
w: *precedent.com/edrichards*
t: @eboyrichards

BENN PEARSON

Benn is a senior frontend developer, bridging the interactive gap between design and development.
w: *precedent.com/bennpearson*
t: @bennpearson

ADAM ELLESTON

Precedent's Drupal lead developer, Adam oversees all Drupal development to maintain excellent standards.
w: *precedent.com/adamelleston*
t: @adamelleston

The RNIB is a leading charity offering information and support to almost two million people in the UK suffering from sight loss. Digital agency Precedent won a tender to create a new site for the charity. The team took a 'form follows function' approach to create a highly accessible, streamlined site that caters to users with varying levels of visual impairment. We spoke to the team to find out how they did it...

What was the original brief?
AE: To design and build a responsive website in Drupal that delivered the highest standards of accessibility, enabling the RNIB to provide more online services and a better online experience.
BP: One of our main challenges was that we needed to adopt and achieve the RNIB's surf right standards: taking the best elements of WCAG across the three levels (A, AA and AAA), while addressing other sight-related accessibility issues for web browsing. We were excited by this, as it offered a chance to push the boundaries of responsive design to improve the web experience for people with sight loss.

How did you begin the project?
ER: The majority of site visitors have varying levels of sight-related difficulties, so in the strategy and UX phases of the project we worked with accessibility consultant Léonie Watson. This enabled us to formulate requirements, test statements and implement training.
BP: Gaining insight into how users interact with the website was a crucial starting point. This enabled us to quickly identify specific accessibility issues, and from there we brainstormed ideas and possible solutions. We found the YouTube videos from blind film critic Tommy Edison a great resource.

How did you approach prototyping?
ER: Once we had formulated the modular approach, we were able to paper prototype all the modules. Stacking the modules made it really efficient and even enabled us to paper prototype the mobile, tablet and desktop views, which fed into how we worked in Photoshop by producing all the modules as Smart Objects.
BP: We then created an interactive prototype. Initially we used this to test responsive modules, but it was then used as an interactive style guide to test accessibility. Finally it became a reference for Drupal integration.
AE: The interactive style guide worked really well during Drupal integration, because it enabled us to quickly implement and test ideas outside of Drupal. It was also a good reference to keep HTML consistent and clean up Drupal's HTML.

Talk to us about colour. How did you pick your palette?
ER: For optimum accessibility you have to have contrasting colours – but we didn't want to replicate the old site, which used solid colour everywhere. We produced a palette from the brand colours and used

*TIMELINE

What happened when during the RNIB site redesign

SEPTEMBER 2013
Precedent enters and wins a competitive tender process.

OCTOBER 2013
Precedent brings in an accessibility consultant, sets up requirements and tests statements.

NOVEMBER 2013
UX and design stage begins, with focus on producing modular elements that can be used anywhere on the site.

DECEMBER 2013
Early adoption of a modular process leads to a similar approach to agile, creating stories focused on modular output to streamline the development process.

JANUARY 2014
Prototype to test responsive modules leads onto an interactive style guide to test accessibility, ready for Drupal integration.

FEBRUARY 2014
Drupal modular approach to integration begins to enable site editors to use modules anywhere on the site.

MARCH 2014
Testing to meet the requirements of people with visual, hearing, mobility and cognitive impairments.

APRIL 2014
Launch followed by lots of press interest and great tweets from @RWD and @drupal.

EVOLUTION

How the team at Precedent created the new site, from initial sketches to final design

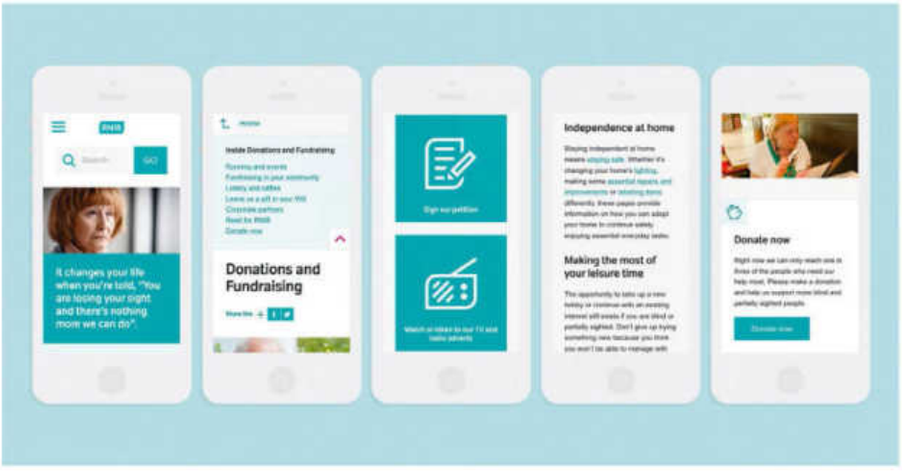
(1) A modular approach was formulated early on, while we were sketching out ideas. **(2)** Initially we looked into producing different templates that worked responsively, but these evolved into responsive modules that only needed one template. **(3)** This approach helped with the later stages of development because modules could be swapped around, added or removed. **(4)** This meant we could produce each module as a Smart Object within Photoshop, which could be pulled into pages and listed in a style guide. **(5)** All of the modules were then worked up into a responsive style guide to test accessibility.

colour contrast tools to meet the WCAG contrast ratios.

What about typography – did you make any clever decisions here?
BP: The typography was part of a wider change in RNIB's branding by The Good Agency (*thegoodagency.co.uk*), but ensuring we were within the WCAG colour contrast ratio formulas was crucial. This was all designed by mathematical formula, and guided our baseline height, which formed the vertical rhythm of the typography.

Is it possible to have a site that's both accessible and beautiful?
ER: Yes, absolutely. The streamlined design we achieved in conjunction with seamless usability is testament to that.
BP: Brand colours, typography and vertical rhythm were all carefully considered, tested and calculated. I think this 'form follows function' approach can lead to sites which are beautiful on the inside and out.

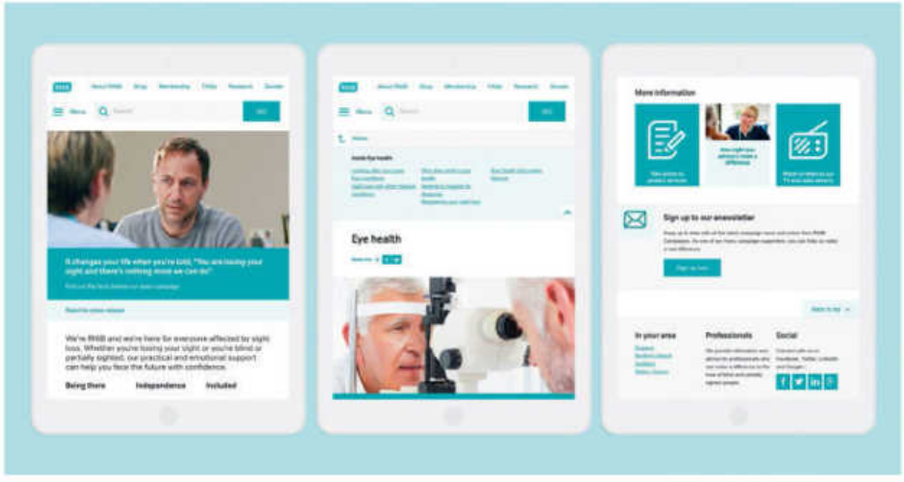
The site is responsive. Did you find this burdensome or liberating?
BP: It can be tricky, but I'm passionate about how a responsive approach can take user experience to another level. In this case it was central to creating a site that's intuitive, streamlined and seamless to use. In development, things such as preventing tab focus on the off-canvas navigation were a little extra work, but responsive was crucial for things like maintaining optimal reading text size and page zooming. Plus, it's the fun bit.



Going mobile The site works on a multitude of platforms - seen here is the mobile version

How did you manage the need to change the scale of the site inside browser windows?
BP: Page zooming works really well on responsive sites, so this helped. The page looks great at 250 per cent taking on the tablet view, and stays consistent as it moves through the responsive views up to 500 per cent where it takes on the mobile view.

Which assistive technologies did you employ and deploy?
BP: Mixture (*mixture.io*) really helped with workflow during the production of the prototype and interactive style guide (*netm.ag/rnibstyle-256*). Using this along with BEM and SMACSS methodologies helped formulate structure and kept code maintainable. Our in-house Sass mixin library was used to make sure everything that was added was kept accessible.
AE: We used Git flow and worked on a feature and release workflow, based on user stories and sprints. We also used Jenkins and Drush to automate releases and make deployments quick and reliable.



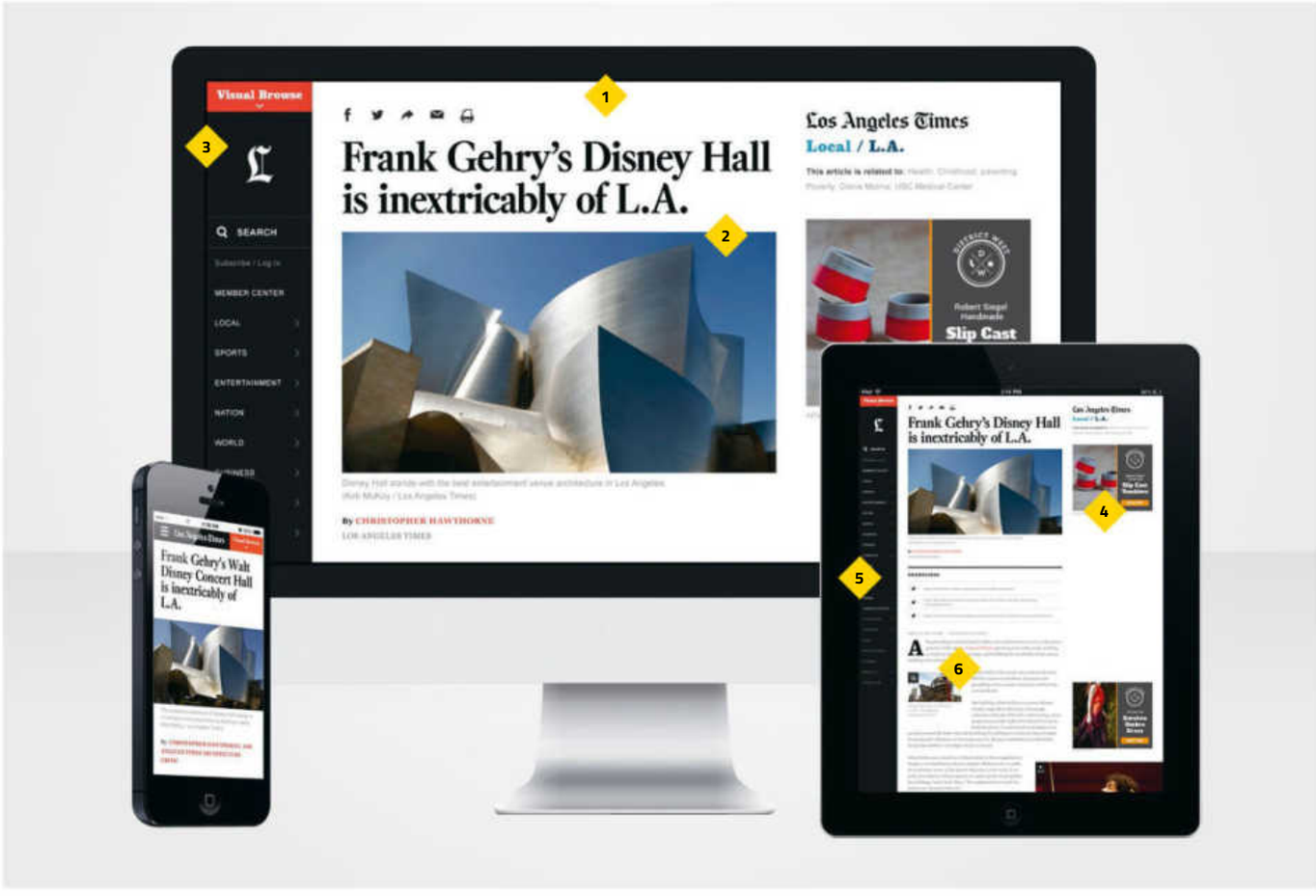
Zooming in Maintaining optimum text size and zooming was crucial, making RWD key in this project

What's the one piece of advice you would give to web designers looking to make their sites more accessible?
BP: Once you've got the HTML semantically structured, without any CSS or JS, your page will be accessible, so just test all the styling you add, and then improve it. One way of doing this is by adding WAI-ARIA, but don't overdo it as it will give users too much information, especially when combined with HTML5 (*netm.ag/info-256*).
AE: Test in screen readers using only the keyboard – just tabbing through the site to see how it's working really helps.
ER: The ambiguous nature of accessibility means that user testing is crucial. Understanding how the users interacted with the RNIB site and formulating solutions was key to the seamless result we've achieved.

HOW WE BUILT

LA TIMES

How **Code and Theory** partnered with the LA Times to create a new, responsive site for one of the USA's biggest newspapers



BRIEF

Los Angeles Times **enlisted Code and Theory to stimulate its thinking about how design and technology can best engage contemporary audiences and serve advertisers, leading to an overhaul of *latimes.com*.**

CLOSE UP

(**1**) The new *latimes.com* was designed and developed to create a singular experience that is visually stunning, easy to read on all devices, future-proofed to accommodate new operating systems and devices, and that helps advertisers reach the *Los Angeles Times*' large and influential audience. (**2**) The 'design API' that powers the new *latimes.com* was designed to manage complexity and amplify creativity within a high-volume, high-velocity news cycle. (**3**) The Visual Browse feature is anchored at the top, providing a photo-centric, section-level way to discover new content. (**4**) The homepage and all section fronts were designed in a flexible format to accommodate the flow of news. (**5**) Slide-out panels offer content discovery and commenting capabilities. (**6**) The Transporter feature introduces a way to path users from one section or piece of content to the next.

DAN GARDNER

Dan is the co-founder and executive creative director at Code and Theory with a focus on user experience
w: *codeandtheory.com*
t: @danjgardner

MIKE TREFF

Mike is managing partner of the Product Design Group at Code and Theory. He previously worked at Soundscreen Design
w: *codeandtheory.com*
t: @codeandtheory

RON PARSONS

Ron is *Los Angeles Times*' vice president of digital product development and a veteran of Buzzmedia and Yahoo
w: *latimes.com*
t: @latimes

With their huge range of editorial coverage, sophistication of ad products and high frequency of updates, newspaper sites pose huge technical challenges for designers and developers – particularly when the paper in question is one of the largest in the United States. Code and Theory and the *LA Times* explain how the fully responsive redesign of *latimes.com* aims to make the newspaper's content accessible on a full range of browsers and devices: past, present and future.

Why don't introduce our non-US readers to the *Los Angeles Times*?
DG: The *LA Times* has an incredibly rich 130-year history. It's a leading source of news and information for Southern California, has won 41 Pulitzer Prizes, and is read all over the world. It's the largest metropolitan daily newspaper in the country – *latimes.com* gets more than 22 million monthly uniques – and has a combined print and online local weekly audience of 4 million. It's a powerful player in the digital news landscape.

What was the redesign brief?
MT: We were tasked to deliver a fully responsive experience for *latimes.com* that not only looked great in any viewport, but worked hard for users regardless of their device and/or circumstance.

Our creative brief was to design an experience that was visually stunning, easy to read and use on all devices, flexible enough to accommodate whatever new devices and storytelling experiences would be needed over time, and would help advertisers better connect with the *Times*' large and influential audience.

The KPIs in the brief were the usual suspects, but upon kickoff we worked together to define many non-traditional KPIs to measure success in potentially more meaningful ways – metrics more aligned to efficiency, user retention, engagement and repeat visitations.

What makes the final site special?
DG: It's one of the largest fully responsive news sites ever built, and it's far more than just a bold new design – it's an entirely new system that better highlights the *Times*' journalism, eliminates clutter, and puts the focus on its unique voices and differentiated coverage. It's flexible enough to adapt to continually evolving news and advertising scenarios and consumption behaviours.

Features like the Transporter, which seamlessly paths users from one piece of content to the next, and Visual Browse, which provides a photo-centric way to discover new content, are just a few of the features that make it so disruptive.

How involved were the journalists?
MT: Very. We embedded ourselves in the newsroom, where we met with key editorial staff, from the managing editor to the *LA Now* editor. The velocity of content they produce every day – breaking news items, long-form investigative pieces, live event coverage, opinion

*LIBRARIES

The team talks though the libraries it used to power the new LA Times site

SITEMESH

A decorator-based framework used to rearrange and combine individual component output in a post-processing step to build the final web page output. It maximises web page performance on dynamic pages by only including code and references to assets needed for the current set of components.

BING MAPS SDK

Provides access by API to the Bing Maps mapping service and map controls. This mapping service is used to display maps of LA neighbourhoods and show points of interest, such as local restaurants, that users can interact with to learn more.

JQUERY

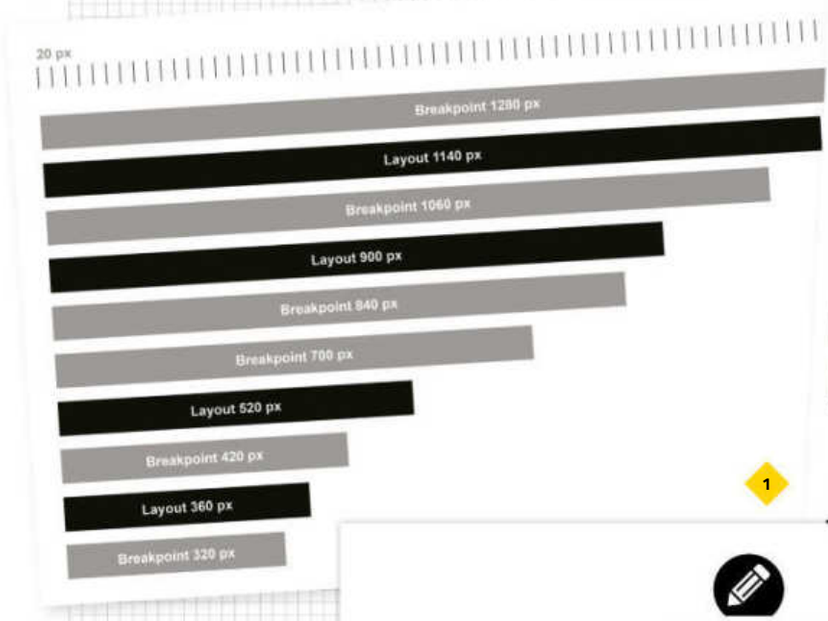
Provides a cross-browser eventing framework and AJAX support. An internal framework builds on event delegation provided by jQuery to power all event-driven functionality throughout the application.

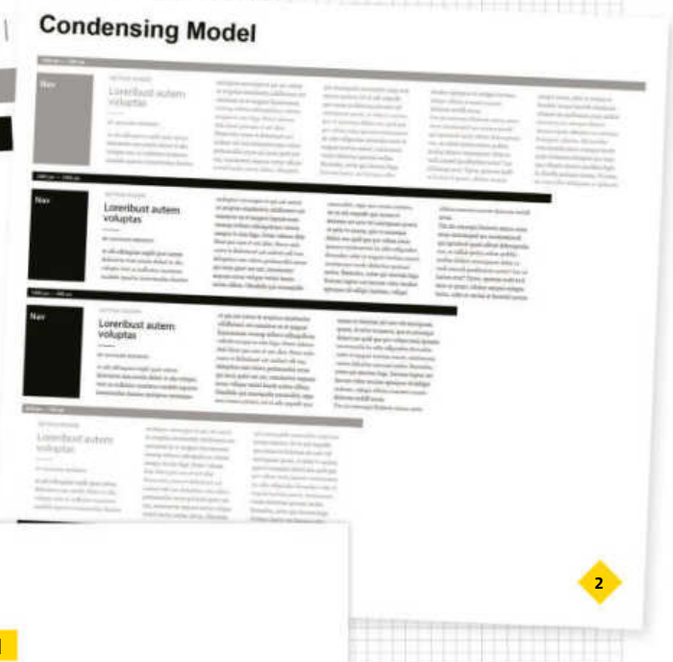
JSTL

A collection of custom tag libraries that implement general-purpose functionality common to web applications, including iteration, manipulation of XML, date formatting and localisation. The main purpose for developing JSTL is to achieve scriptlet free JSP.

MEMCACHED

Provides a shared, distributed memory cache for arbitrary information. This allows static or immutable content to be retrieved from the database or expensive web service calls once, and then requested many times without incurring additional overhead.

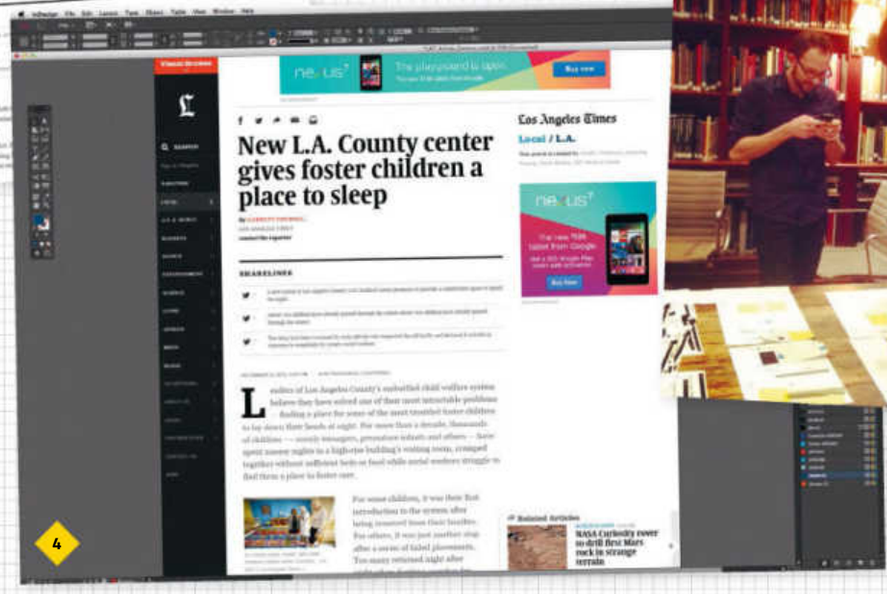




EVOLUTION

Code and Theory explains the stages of its design process during the responsive redesign of *latimes.com*

(1) Rather than using breakpoints targeted to the screen sizes of specific devices, the site was designed to midpoints. This grid can expand or shrink to every screen size, even for devices that haven't launched yet. **(2)** The starting point was the smallest, most condensed version of the page. When screen size increases, the layout expands in wider blocks. **(3)** Initial designs focused on the article page. **(4)** InDesign is known as a print design platform, but is also equipped for scalable responsive web design. **(5)** To test the new system, paper prototypes of every possible outfit were spread out on four tables in the Code and Theory library.



columns, reviews, photojournalism and video – is mind-boggling.

What about advertising?
MT: The *Times* knew its advertisers needed a more appealing visual landscape that would draw more readers to the page. It had to be less cluttered to make ads more noticeable and effective.

We broke apart the traditional page grid to solve for 'right-rail blindness', and designed new fully responsive, large-format ad units that would offer advertisers an execution as technologically advanced and visually arresting as the rest of the new site.

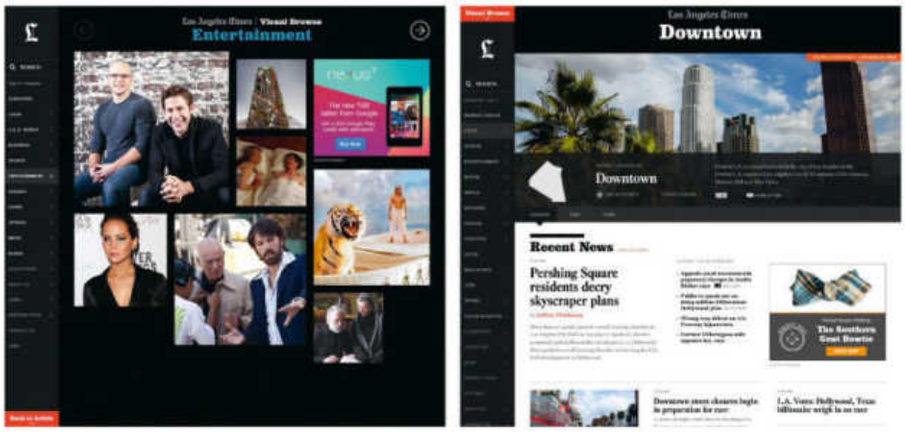
When building the grid system, we considered every possible scenario – for instance, how the system would respond if a homepage takeover was in place while a breaking news event was occurring.

What was the biggest technical challenge you encountered?
RP: Coding the Transporter (our version of the endless scroll). Getting it to perform well across different browsers and mobile devices was a very big challenge. Working with browser history APIs – which we had to – in order to enable the type of forward and backward scrolling the spec called for posed a unique set of challenges.

The site looks lovely on new phones. But what about legacy devices?
MT: Given the size and breadth of the user base, performance was at the heart of many of our design decisions. We wanted an experience that didn't require degradation, but that would be utilitarian. In addition to making sure legacy devices were accounted for, we tried very hard to future-proof the experience.

We built a flexible grid that adapts a wide variety of currently unforeseen viewport spans. Rather than creating a responsive design with specific breakpoints targeted to the screen sizes of certain devices, we designed the grid around this question: "What is the smallest unit of legible text and how is it laid out on the page?"

Our design methodology allows for easy development of new responsive modules to meet evolving storytelling, content promotion and advertiser needs.

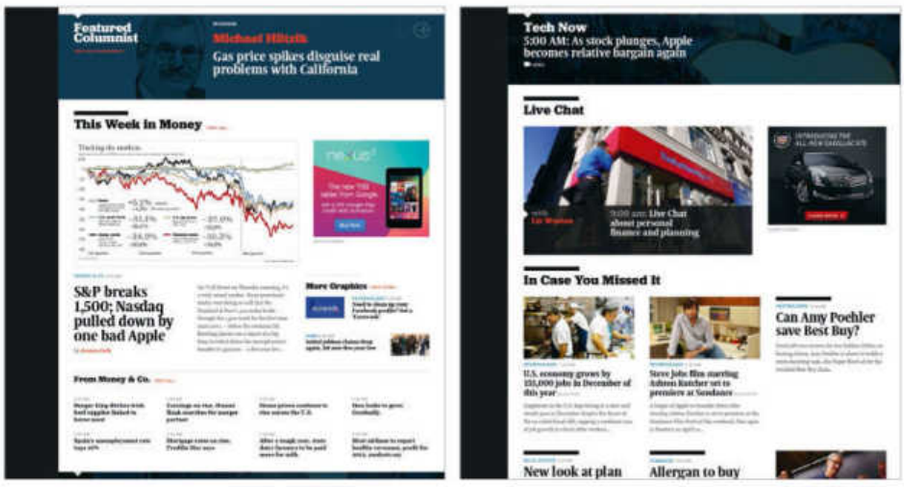


Left The Visual Browse feature provides a photo-centric way to discover new content **Right** Neighborhoods is a 'hyperlocal' feature pulling in neighbourhood-level geocoded news and information

What about users with slow or intermittent internet connections?
RP: We didn't design for offline per se, but we did focus on page speed and overall performance by employing heavy caching and delayed loading techniques in our JavaScript frontend rendering framework. We cut homepage load times significantly and sped up rendering across the board.

What's the public reaction been like?
DG: The industry response has been great. The famous publication designer Dr. Mario Garcia called it "exquisite", and even journalists like Neela Banerjee (a self-proclaimed cynic) said that it rocked. And it's clear that readers are engaging more with the site: Just one month after launch, the *Times* saw a 66 per cent increase in page views per visit.

What was the biggest lesson you learned along the way?
RP: Supporting sophisticated interactions in a responsive framework, across as many browsers and devices as we did, is a gigantic task and requires compromises on design, product, and technology. But the reward can be worth the risk.
MT: We realised early on that the print product – which features different layouts almost every day to reflect the news cycle – was actually more flexible than most digital news experiences. That was shocking to us: the physical edition was more flexible than the digital, which is nothing but code.

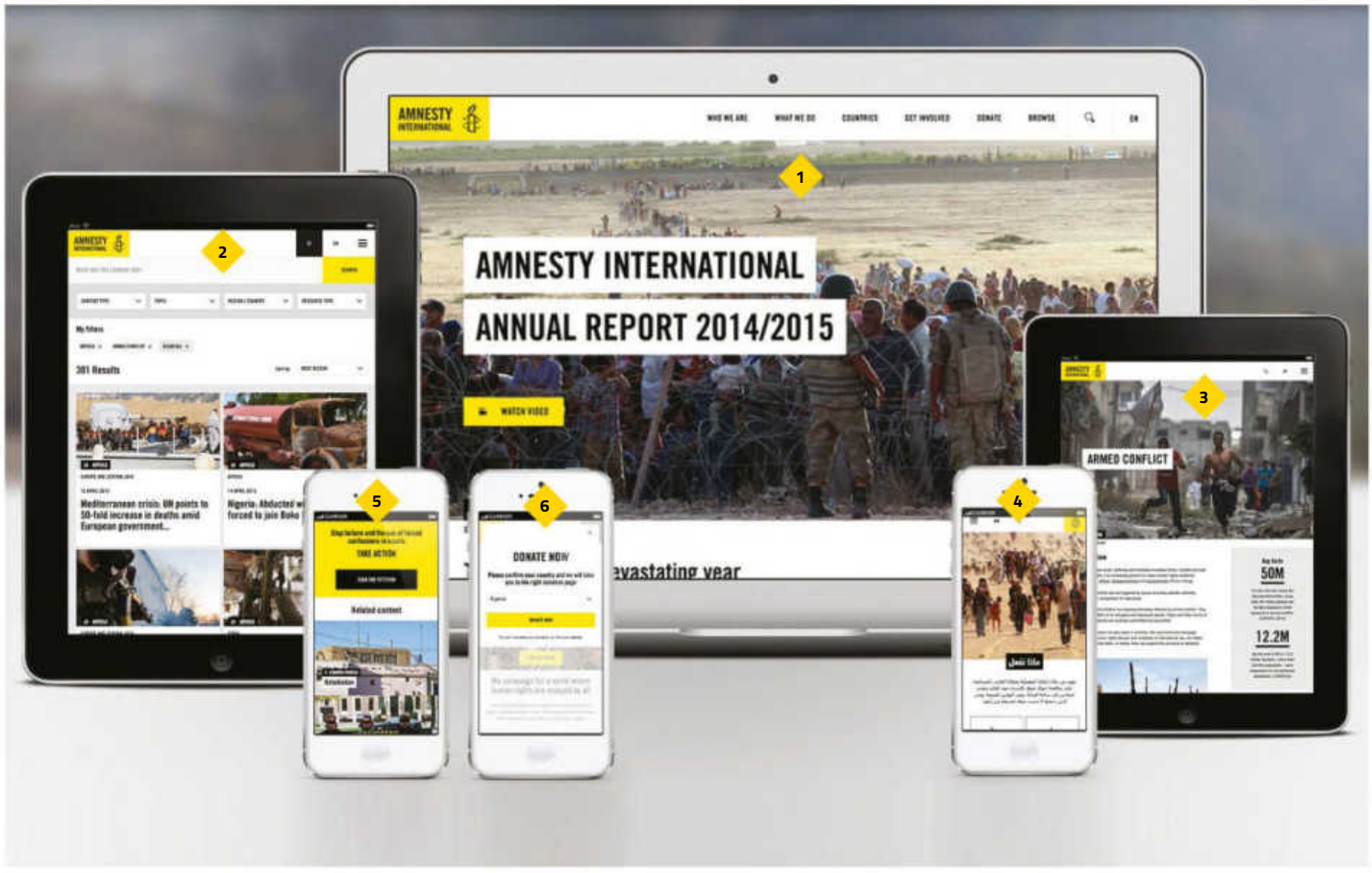


Clutter-free The design of the site is intended to allow users to consume content with minimal distraction

HOW WE BUILT

AMNESTY INTERNATIONAL

Code Computerlove used compelling storytelling and stunning imagery to encourage users around the globe to take human rights personally



BRIEF

Code Computerlove was asked to create a new platform for human rights organisation Amnesty International. This would act as the global face of the charity, enabling it to engage with audiences around the world and encourage human rights change.

CLOSE UP

(1) The new site uses storytelling combined with photojournalism and editorial content for maximum impact. **(2)** With a best practice global search and filter function (plus news feed), users can easily get to the information they need and that is relevant to them. **(3)** Every country has its own page, featuring a compelling, localised story of the state of human rights in that area. Strong imagery and components like 'Quick Facts' help viewers take in information at a glance. **(4)** The site design needed to work across multiple different languages – the layout flips in reverse when viewing in Arabic. **(5)** Every page has tailored onward journeys for either key conversion actions like 'sign a petition' or to read more content. This ensures there are no dead ends. **(6)** We made it extremely easy for any user to be able to donate to their chosen country at any point on the site.

COLIN PRESTON

Colin is a creative lead at Code Computerlove and directed the site design for the Amnesty International project
t: @colinpreston

SALLY ANDERSON

An account director at Code Computerlove, Sally led the implementation of the new Amnesty platform
t: @sallyeandersons

LUCY ANDERSON

Lucy is Code Computerlove's research and insight manager, and led customer research on the Amnesty project
t: @LucyDaisy

Amnesty International is a global movement that campaigns to end abuses of human rights. Its new platform, designed and built by Code Computerlove (*codecomputerlove.com*) sits at the heart of the organisation's digital framework. The site, which is led by bold imagery and powerful storytelling, aims to inform users around the world of Amnesty's work, and encourages them to get involved.

Can you introduce the new Amnesty International site?
SA: *amnesty.org* is the public face of Amnesty International. It not only communicates what Amnesty does, but also inspires people to get involved.
LA: A key feature is the use of multimedia content to deliver an immersive online experience. User journeys have been carefully crafted following in-depth research and testing. The site is multilingual and has been designed to appeal to a wide variety of users who may be visiting from anywhere in the world, and for a variety of different reasons.
CP: A key challenge was to define the global site's role in comparison to Amnesty's local sites, such as Amnesty UK. It had to complement these existing local sites and also cater for countries that didn't have a local site.

How did you get involved with the project?
SA: We already have strong charity credentials thanks to our ongoing relationships with organisations like Greenpeace, Sue Ryder and WaterAid. Amnesty approached us after seeing what we'd achieved for Oxfam. We worked on a consultancy exercise to define Amnesty's requirements and strategy and then won the pitch for the design and build.

What problems did the redesign look to solve?
LA: The brief was to create a platform that catered for a worldwide audience, and that would sit at the heart of Amnesty's digital framework. The platform needed to act as the destination gateway for people to take more action by volunteering, involving themselves in a campaign, signing petitions or donating. It was also important to target a younger audience who aren't as aware of the charity.
CP: Another objective was to showcase the best campaigning from around the world, and the diverse nature of Amnesty's work – for example, how global issues like 'Stop Torture' are being addressed very differently in the US to India – in order to highlight local issues and appeal to different markets.

Amnesty International shines a light on human rights issues through its research and journalism. The project looked to make this content more accessible and bring real-life stories to the fore in order to encourage people to take human rights personally.

Given the gravitas of Amnesty's work, did this project feel different?
SA: We have lots of experience working with large-scale global charities and know

TIMELINE

How the project progressed, from brief to completion

JUNE 2014
Strategy phase, including audience and stakeholder interviews to help define the role and scope of the new site. Presentation of the site vision

AUGUST 2014
Following a pitch process, Code Computerlove is appointed for the design and build

SEPTEMBER 2014
Kickoff of the experience design phase, working with Amnesty's design and UX team to develop the initial visual style

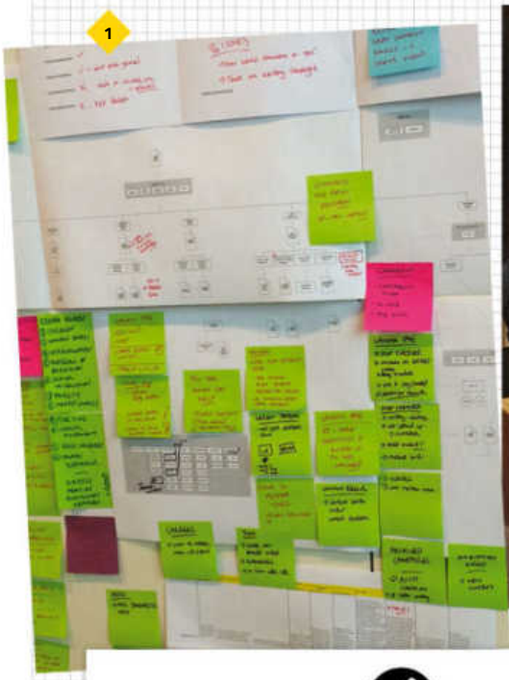
EARLY OCTOBER 2014
User testing of priority journeys, and refinement of wireframes, designs and user stories based on feedback

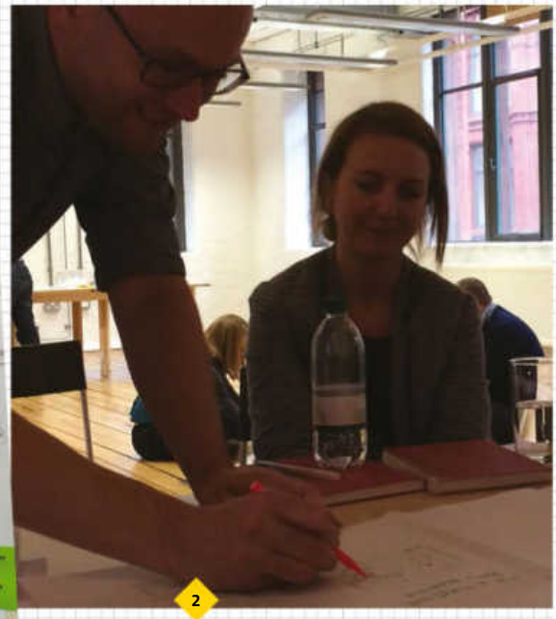
MID OCTOBER 2014
Content strategy implementation including performance review of old site content against new strategy

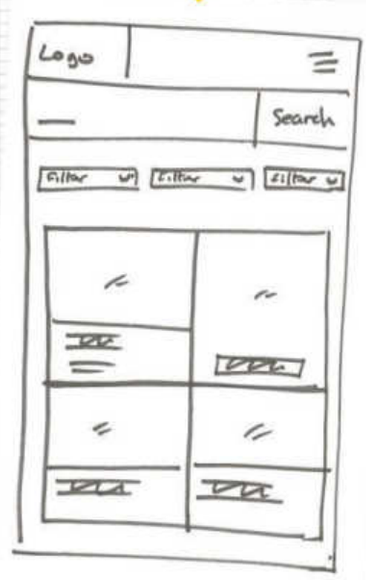
LATE OCTOBER 2014
Start of development phase, including set-up of Umbraco CMS and Azure hosting environment

NOVEMBER 2014
Development sprints commence, focusing on key user journeys. Released for testing by the client every two weeks, with load testing built into each release

FEBRUARY 2015
Initial launch of new responsive site to coincide with the launch of the Amnesty Annual Report on the status of human rights. Development continues on additional functionality



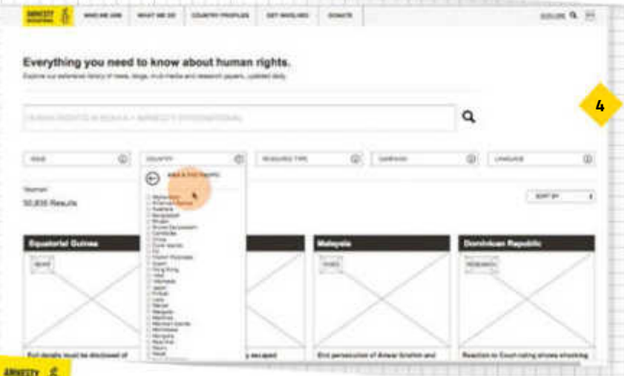


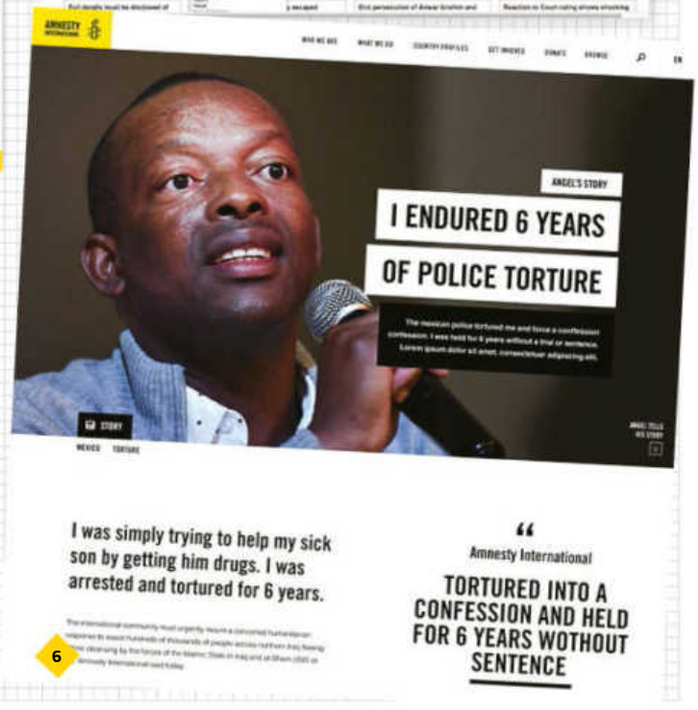


EVOLUTION

A look at how the visual concept developed as the project progressed

(1) During the solution design workshop, we discussed each page's features and content. **(2)** We sketched out ideas in groups, and collectively scored the ideas we liked. **(3)** This quick sketch for a medium screen size shows the search page, detailing the filter positions and grid of results. **(4)** Wireframe prototypes helped us test the search page features and taxonomy. **(5)** More detailed concept sketches were used to nail down the look and feel of the site. This sketch shows the story page template. **(6)** Finally, we started designing pages in more detail.

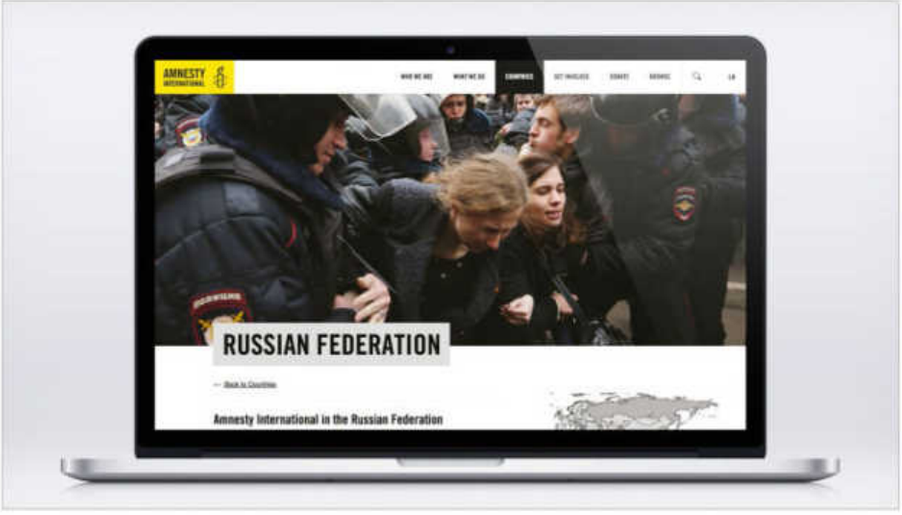




how import digital is in achieving their objectives. We started working on the project just after news broke of the disappearance of a number of girls in Nigeria under Boko Haram, and this really brought home the importance of Amnesty International's work.

LA: In developing the strategy we spoke to a number of people affected by human rights issues. Being able to hear first-hand about the impact that Amnesty's work has undoubtedly helped us go the extra mile.

CP: As a designer I always aim to solve problems with best-in-class solutions, but during the early stages of the project I became very aware of just how important the site was to people in all walks of life. I felt a huge personal responsibility to ensure the design stayed true to Amnesty International's values and vision.



International issues The site works to surface global issues alongside regional content

How did you test how effective the site was at meeting objectives?

CP: Because we needed to engage users that spoke different languages, came from different cultures and possessed varying levels of technical aptitude, user testing was paramount. It allowed us to test out any perceptions of how people would use the website.

LA: It's not good enough to simply produce a nice-looking site and hope it will perform against the client objectives. A/B testing ensured we launched with the most effective content and we are continuing to further optimise the pages post-launch, as we learn from how users are interacting with the site.

CP: The creative team here worked collaboratively with Amnesty's UX team to help deliver a top-performing site. We undertook numerous whole-team workshops involving frontend and UX, sketching out pages with persuasion techniques built in to aid conversion.

How important was prototyping?

SA: Prototyping allowed us to test some key interactions before we got too far into production. Donate, join and search journeys were created as HTML prototypes, which evolved throughout the build. The site is in four languages (including Arabic, which runs right to left) so another key area we prototyped was whether the designs would work across all languages.

CP: As this was a mobile-first site, we focused particularly on the search function. We quickly found out whether certain interactions felt natural or worked on the small screen. This was much more efficient than designing screens in Photoshop and explaining the interaction to a developer.

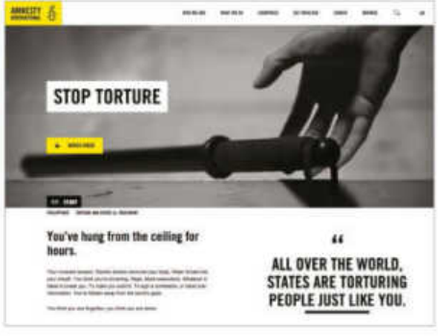
Did you talk to content producers during the design phase?

LA: Absolutely – this project was very much approached content-first. We gathered insights from key content producers across the globe through telephone interviews and online surveys. And we didn't stop at writers; we also got input from researchers, human rights activists, teachers and campaigners.

CP: We prototyped with real content to check the designs would work mobile-first. This ensured we championed content from the start and helped reduce common responsive snags.

Can you walk us through your approach to storytelling?

CP: The quality of Amnesty's existing content and the nature of its work meant storytelling became a key part of the strategy. The organisation has some amazing photography, which is especially powerful when coupled with the emotive stories from people in need. We recognised early on that from a creative point of view we wanted to let these things shine. We developed a story template that mixes shortform and longform, along with different media to keep the pages rich in content.

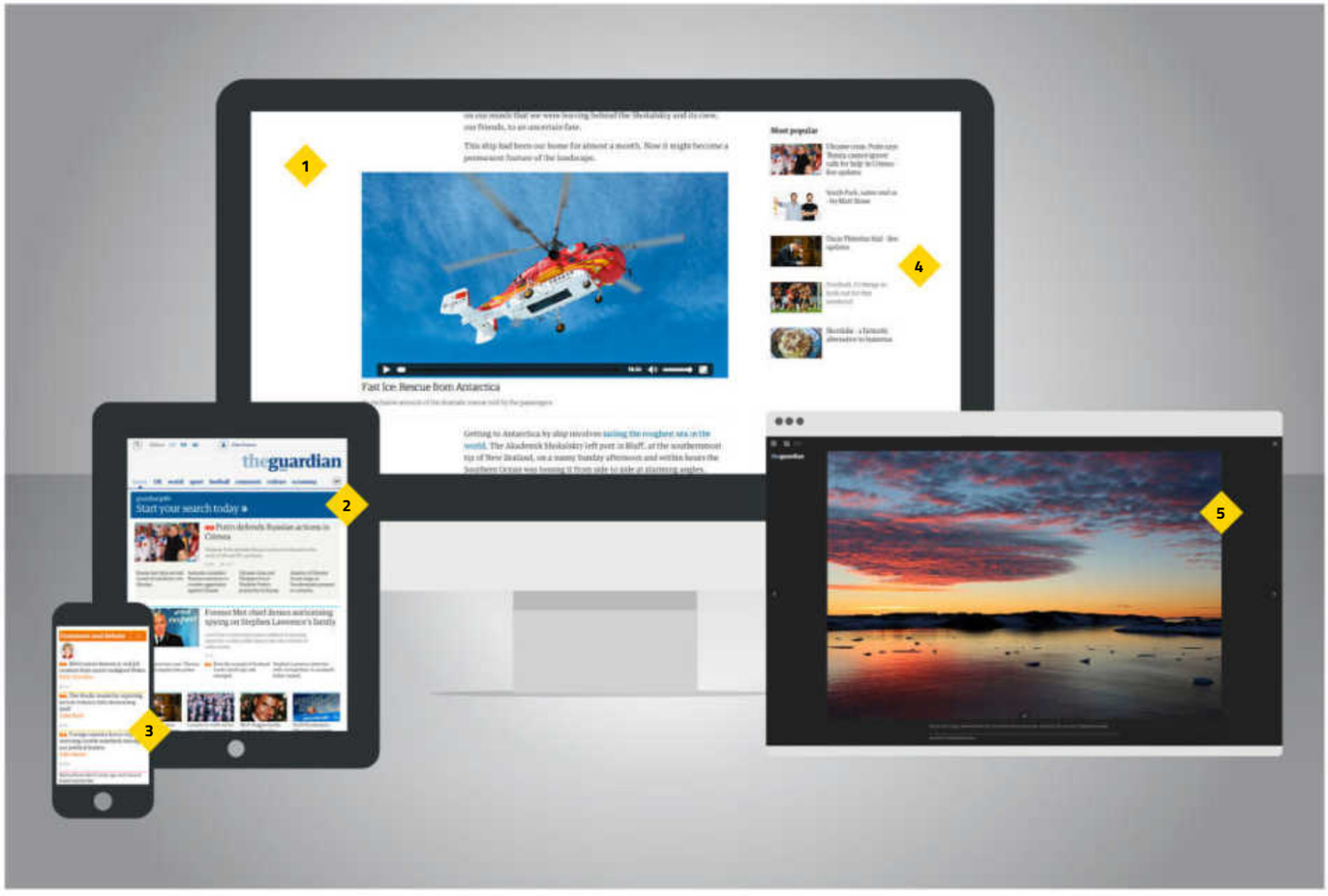


Making an impact Amnesty's strong emotive photography encourages users to take action

HOW WE BUILT

THE GUARDIAN

Developed by its in-house team, the **Guardian**'s new responsive site presents news to mobile devices in a clean, flexible format



BRIEF

With the continued migration of its audience to mobile and tablet, the Guardian newspaper rebuilt its site to make it easier for users to discover a wider range of content, and to better integrate display ads.

CLOSE UP

(**1**) The Guardian's new responsive website (*www.theguardian.com*) is designed to provide a better user experience on mobile devices. Part of this brief was to make it easier to expose feature-sized embedded content to mobile and tablet users. The CMS now allows for huge flexibility and variety within templates. (**2**) One of the aims of the new design was to make it easier for mobile visitors to find a range of the newspaper's online content. The homepage containers are built to be adaptable in order to reflect the news of the day. (**3**) The visual language reflects the tone and voice of the content throughout the site, from section pages to content pages. (**4**) A personalised recommendation system ensures onward journeys are more relevant to the particular user. (**5**) Rich imagery is allowed to shine through full-screen experiences, such as in this immersive gallery.

NICK HALEY

Nick is the Guardian's director of user experience, and a 'dad, cyclist and whiskey drinker'
w: *theguardian.com*
t: @twobobswerver

PATRICK HAMANN

Patrick is senior client-side developer. He is passionate about frontend performance
w: *theguardian.com*
t: @patrickhamann

One of the world's most widely read newspaper websites, the Guardian (*www.theguardian.com*) attracts tens of millions of unique visitors each year. With that audience increasingly migrating to mobile and tablet, the newspaper's in-house team was tasked with rebuilding the site to improve user experience on mobile devices. As well as making it easier for a mobile audience to discover a wider range of its content, a key objective of the new site was to better integrate display advertising and other commercial offers. Below, the Guardian's team tells us about their workflow, maximising performance on such a widely used site – and just what the hell 'swimlaning' means.

What are the challenges of responsive design on such a large scale?
PH: The hardest thing we have found is changing our internal tooling. We've had to create new content-management tools that allow us to create flexible content that can be displayed across many devices. Most importantly, we have created a shared vocabulary between designers and developers, including typographic scale, colour palettes and a module system. We call this Guss (*github.com/guardian/guss*).

How has the information architecture of the site changed?
NH: One of the first things we did was look at how we could simplify our information architecture. We saw from analysing user data that once a user gets to a certain point in their journey they stop using the navigation and instead navigate through content. The old site had multiple levels of depth, and subsections would often have bespoke navigation that was out-of-date, cluttered and poorly used. We've stripped back the navigation to two levels: primary and secondary. We also changed some of the groupings we use to categorise our content, which we validated with several card sorts. This allows people to quickly orientate themselves to the main topic area they're interested in, and dive straight into content from there.

How did the content strategy evolve?
NH: A key part of the strategy for the new site is better discoverability. We publish over 500 articles a day, and we want to make it easy for people to find them. One of the things we're experimenting with at the moment is the idea of blended content. Rather than a traditional classification system, with homepage areas for news, sport, technology, culture and so on, we've been testing a 'people' zone and a 'reviews' zone which aggregate content from the whole of the Guardian. Rather than pigeonholing content by how it was commissioned, blending frees it to appear anywhere. We're really excited about it.

What was your wireframing process?
NH: One of the things we did right from the start on the project was to wireframe in HTML rather than using traditional UX software like Axure or OmniGraffle. For a responsive design project, this has

*CHRONOLOGY

The critical path of a responsive redesign

PROJECT INITIATION
Mobile is becoming the site's fastest-growing platform. The decision is made to go responsive and build in-house

STEERING GROUP
An editorial steering group is established to guide content strategy

PROTOTYPING
User testing of first prototypes begins

ALPHA BUILD
An initial alpha is launched to external readers after a few months

MOBILE SWITCHOVER
All mobile readers are switched to the new responsive platform. Engagement across the board increases

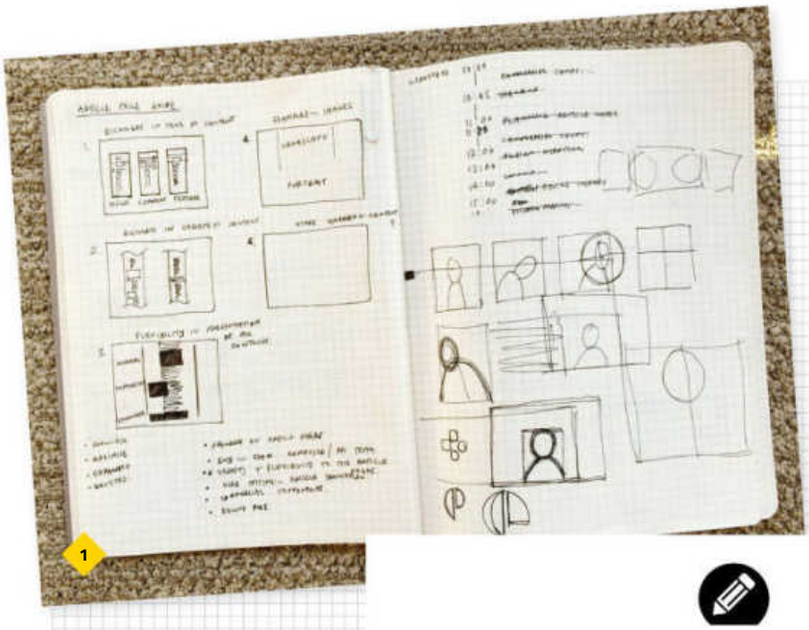
INTEGRATION
Integration of existing Guardian products such as commenting begins

TOOL DEVELOPMENT
New internal tools are built to power the curation of content across all viewports and contexts

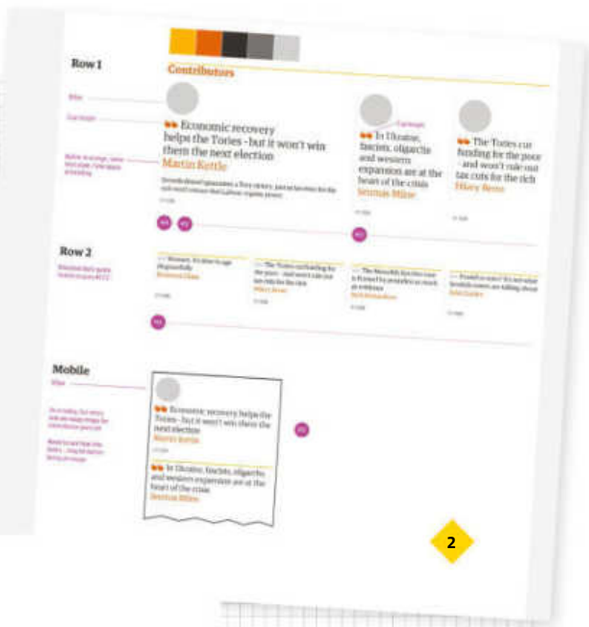
ENRICHMENT
New richer interactive templates, galleries and videos are added

BETA LAUNCH
The beta site is launched across all viewports to an audience segment

GLOBAL ROLLOUT
The responsive site is rolled ou to all Guardian readers



1



2

EVOLUTION

The Guardian team explains how the new site evolved, from sketches through implementation to final testing

(1) Sketching remains the most immediate way for communicating ideas throughout the design process. **(2)** Sketches for content containers, translated to digital form. **(3)** A team pinboard. We built up a visual language based around tone of content. **(4)** The changeability and the huge volume of content has been a big challenge. Getting concepts built in code has been the only real way of tackling this. **(5)** We have been testing every step of the way: both multivariant tests with real data and lab tests.

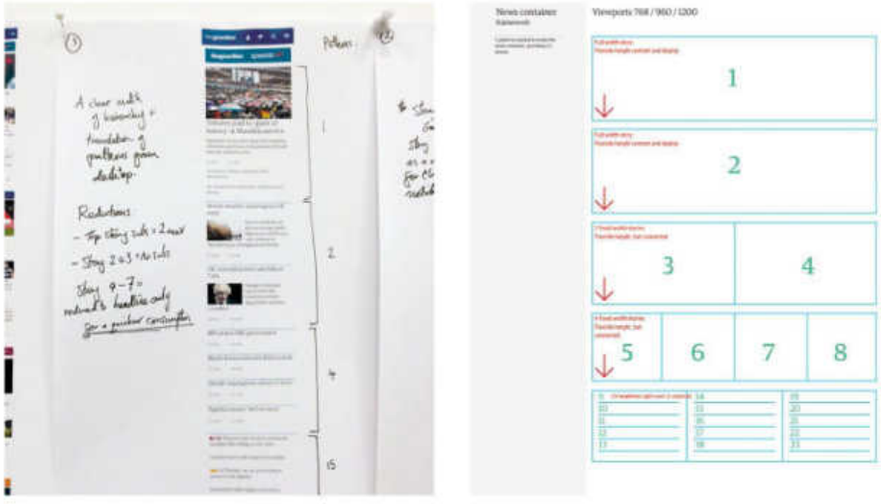
Designers Chris Pearson and Kat Stubbings created images for this article

3

4

5

a number of benefits, the strongest being that you can immediately see how an idea will work in a browser and iterate from there. HTML wireframes are also incredibly easy to share with others. Rather than using annotations that no one ever reads to describe an interaction, it's right there on the page.



Left The new design makes it easier for smartphone and tablet users to navigate to the content they want
Right The homepage containers are built to be adaptable in order to reflect the news of the day

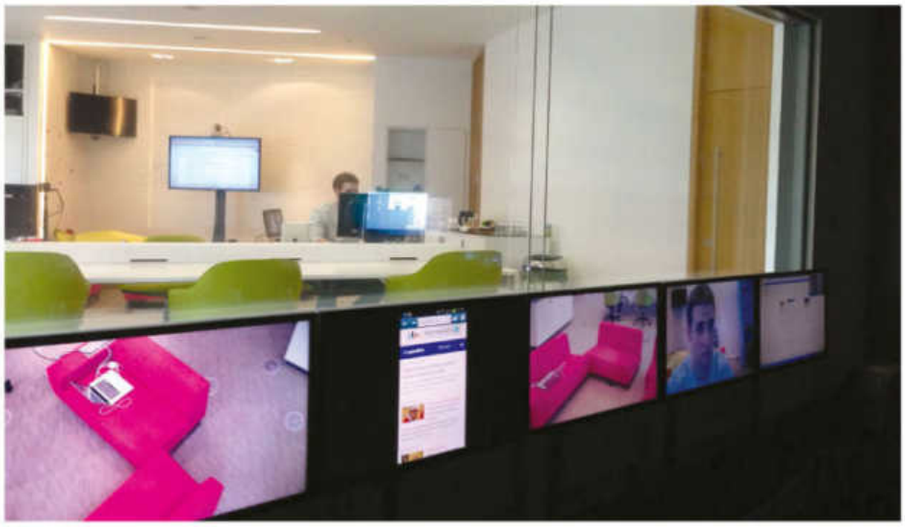
Can you give us an outline of your technology stack?
PH: On the server side we're using the Scala flavour of the Play MVC framework to power the site. On the client side we harness the power of Sass to write our CSS and keep our JavaScript modular with AMD. Finally we use the Node.js task runner Grunt to tie this all together into a compiled application.

There has been a lot of talk of 'swimlaning' on the project. What is it?
PH: One of the biggest lessons we have learnt from 15 years of software development at the Guardian is that large monolithic applications are a nightmare to maintain. With that in mind we have spent the last three years separating our entire infrastructure into a service-orientated architecture.

Our fantastic content API (*netm.ag/guardianapi-254*) means our frontend application never has to query a database and is purely responsible for rendering content. We take this notion of separation one step further with swimlaning (*netm.ag/defineswimlaning-254*): decoupling parts of the rendering tier so that they can be hosted and updated independently. For example, our articles are served via different servers to our galleries – so if our gallery server crashes and burns, access to articles won't be impeded.

What techniques have you used to maximise performance?
PH: Latency is the biggest performance killer, especially on mobile. The easiest trick is to reduce HTTP requests and round trips. We do this by utilising the browser's localStorage, caching everything from web fonts to CSS in there.

In turn, this prevents requests on subsequent pages. We also inline our critical CSS into the initial HTML payload to trigger instant rendering. All of these techniques make the site more resilient to failure.



Test suite The design was evaluated as it evolved, including regular face-to-face sessions with users

Was performance in the minds of the whole team, or just the developers?
PH: Performance needs to be everyone's responsibility. To do this, we've had to educate our designers and product owners about the benefits. Firstly, you must quantify the impact of speed on user engagement, and then invest in tools which can visualise the site's performance in real time. We have dashboards that are always visible to the team giving constant feedback so we know instantly if a change we shipped has slowed something down.

How often do you deploy updates?
PH: We strive to continuously deliver change to our users, whilst developing and testing in the open. We now deploy to production on average four times a day, and have even been known to do so from an airport in Jersey – or even live on stage from conferences.

net PRO

YOU ARE INVITED TO JOIN NET PRO!

For more than 20 years, net magazine has been at the forefront of web design and development. Join **net Pro** and as a **special introductory offer, you can save £30!**

WHAT YOU'LL GET

- 13 issues of net in print and digital
- Annual Industry Report worth £100
- Discounts to industry awards including Generate
- Inspirational design annual worth £9.99
- Monthly net Pro email newsletter
- Special discounts from selected design industry partners

ALL FROM JUST ~~£135~~ £105

WORTH £371

JOIN TODAY VISIT www.myfavouritemagazines.co.uk/netpro

Terms and Conditions: Pricing and savings quoted are compared to buying full priced UK print and digital issues. You'll receive 13 issues in a year. Prices correct at point of print and subject to change. Full terms and conditions *myfavm_ag/magterms*. Offer ends 31 December 2016

FOCUS ON

MESSAGE VS MEDIUM

Designing responsively isn't always enough – we need to focus on how we deliver content for different devices, says **Gene Crawford**

There is an inherent issue with the mobile-first approach to creating websites. That issue is that when we design for the smallest screen size first and scale things up, often we wind up with designs that are boring or 'dumb' when they reach desktop screen sizes.

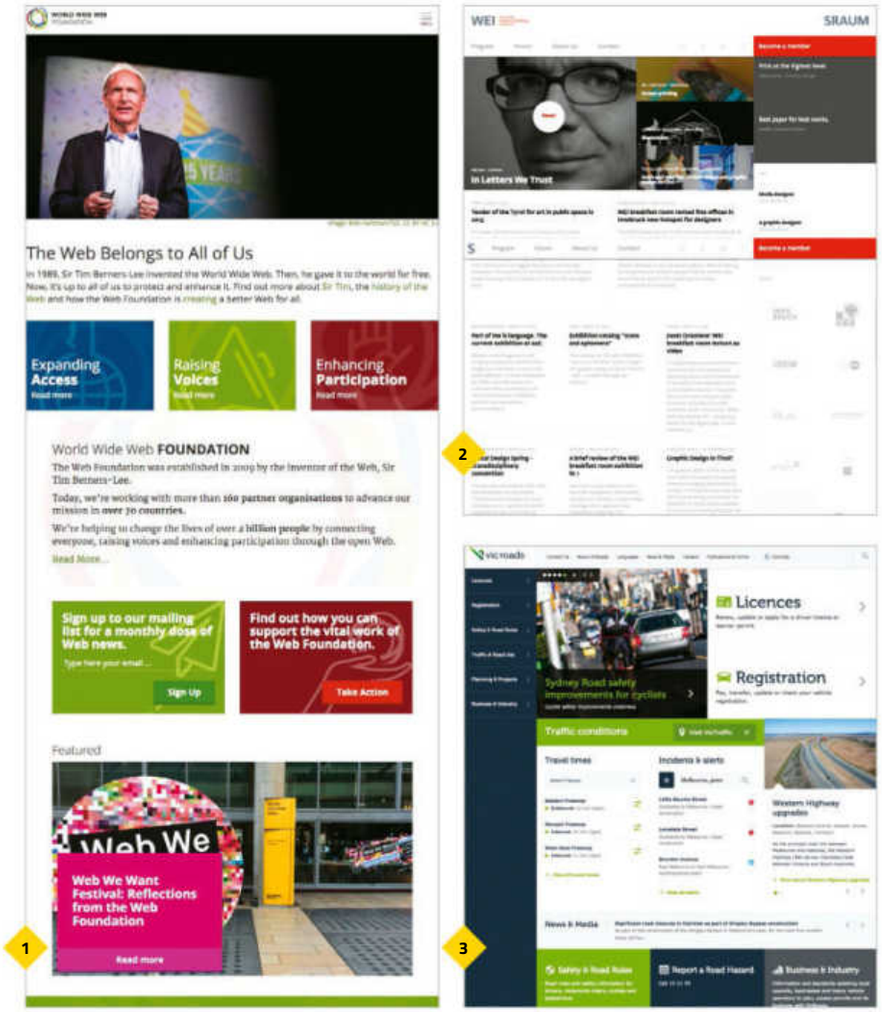
The aim of RWD is to deliver the same content, built once, to a vast array of screen sizes. Essentially taking the same content and delivering it via a flexible grid, positioned vertically into a pre-arranged order. As the screens get wider the grid expands, and that same content is arranged in a different order. Assuming we place the most relevant stuff closest to the top of the screen, we win. Right?

What we're doing is basically taking bets on what's important to a user on mobile as opposed to desktop. It comes down to how good the designer is at figuring all that out up-front. Often the larger screen widths lose out by just being blown up versions of the mobile-first design. This likely has added to the explosion of 'flat design' that we've experienced for the past couple of years. Not to mention all the various mobile navigation patterns we've seen.

We have to take into consideration the medium we're designing for. Responsive design is the standard through which we deliver the message, but it's not always enough to simply combine that with a mobile-first approach. I don't know where we're headed, but we must focus our efforts on fine-tuning our delivery.

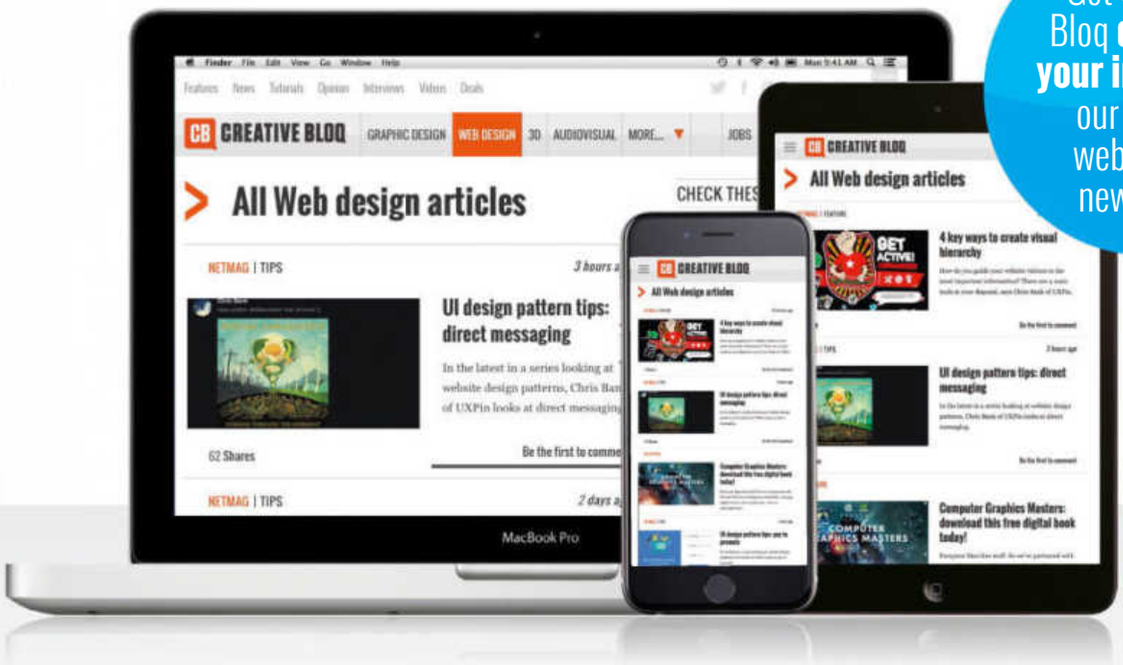
PROFILE

Gene's mission is to work tirelessly to provide inspiration and insight for developers. His recent projects include *unmatchedstyle.com*



(**1**) The Web Foundation's site (*webfoundation.org*) is a beautifully executed responsive design. However, it places an image as the main element on the smartphone version. (**2**) The site fordesign learning programme WEI (*weissraum.at*) takes into consideration the various scales of the devices through which users might experience it. (**3**) The site for VicRoads (*vicroads.vic.gov.au*), the Australian government motor vehicle service, keeps the two most important things that users are likely to need front-and-centre, no matter which screen width they visit it on.

The number one destination for **web design** news, views and how-tos.



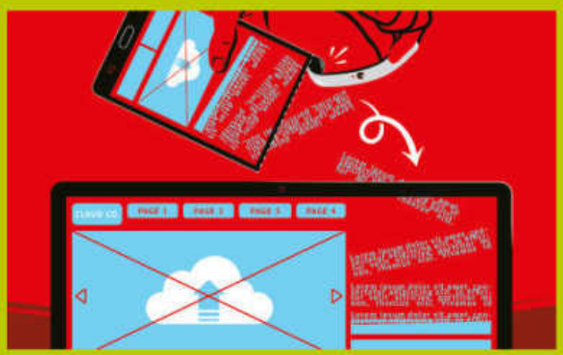
Get Creative Bloq **direct to your inbox** with our weekly web design newsletter

CB CREATIVE BLOQ

Graphic design | **Web design** | 3D | Digital art

www.creativebloq.com

VOICES



RWD

GIVE SITES BACK THEIR SOUL

Illustration by Ben Mounsey

As responsive design becomes the norm, designers have developed a reliance on boxes and grids. **Noah Stokes** considers how we can put the magic back in our sites

About a year and a half ago, I had seen enough. A tweet came through my stream and like so many others at the time, it was a link to a hot, new, amazing #RWD site. 'Oh, I love an amazing design,' I thought, so I clicked. What I saw was a design casualty: boxes and grids everywhere. This was the 'amazing' new design? To me it looked like every other #RWD out there.

So I did what most people would do; I composed a tweet. "I hate #RWD."

That may not come off right, I thought. So I did a little re-composing. "I really hate #RWD."

That's more like it. It's blasphemy, right? RWD is the future of web development. The thing was, to me, the web was losing its magic. And I blamed RWD. In order for me to explain why, we have to go back to the web of the early 2000s.

In those days, websites called K10K, CSS Zen Garden, Stylegala and 2advanced were some of the biggest sources of website design inspiration out there. Stylegala would feature a Site of the Month that, for me, was like Christmas morning to a six year old. I would spend hours exploring these sites, studying all the wonderful details; all the passion the designer had put into creating it.

Back to my tweet. I had enough wits about me to completely rewrite it to something less trolly and more 'politically correct'. "I feel like responsive design has sucked the soul out of website design. Everything is boxes and grids. Where has all the creativity gone?"

Away it went. Someone created a branch around it that went on to get close to 15,000 views. A lot of folks (rightly so) were asking me to define what a 'soul' was. This all goes back to my earlier references to K10K, Stylegala and so on. Those sites featured designs that were so unique; so full of thoughtfulness at every turn. My early exposure to web design was full of soul. I would say a soul is the intangible details of a design.

You've seen sites like this. Sites that stand out from the others. What sets those apart? To the untrained eye, they look like any other site, but to those of us that love design, these sites have a soul. I miss them. Why don't we see many of them around anymore? I think I have a pretty good idea why.

DREAMING TIME

Elliot Jay Stocks wrote an article called 'Why have today's designers stopped dreaming?' (*netm.ag/dream-263*). In it, he said:

"You click the website link and the assets begin to load. The page is a gigantic photo, filling the entirety of the viewport. In the photo is a group of people ... the product is just about visible on the screen of one of their iPhones. Overlaid on top of this photo is a brief sentence that introduces the product in a not-really-saying-anything-in-particular sort of way, set in a sans-serif typeface ... You're gently transitioned further down the page and the large, cheerful photo remains in place, without scrolling, until it's obscured by the main content area

that appears from below ... You scroll a little further down the page and small diagrams begin to animate themselves as they enter the viewport. Then ... you notice that a semi-transparent menu has slunk into view and attached itself to the top of the page. It stays there as you scroll through the product descriptions, the giant bands of flat colours and the well-lit photos of the support team pulling their best 'fun' faces."

Sound like a website you've seen before? Trends come with the business, and they can be something to embrace. I love what Stocks goes on to say though – just because we have this style that is very minimal and subdued doesn't mean that we shouldn't push ourselves as designers to create beautiful experiences online.

RESPONSIVE QUESTIONS

What about RWD? There's no doubt that RWD takes longer to develop than those old fixed-width sites of my past. Chris Cashdollar of Happy Cog wrote a great article about how RWD has kind of thrown a wrench into their design process (*netm.ag/cashdollar-263*).

When Happy Cog would present a design to a client, that client would inevitably ask what it would look like on mobile. For Cashdollar's team to create a second or third set of comps just for mobile was putting a strain on an already stretched budget. So instead of focusing on beautifying pixels, they took a different approach: "We needed to cash in all those hours spent perfecting pixels on a process with a higher return on investment." Happy Cog found itself letting go of design flourishes – or soul – in order to meet its client budgets."

OK, so trends and budgets can be a factor. What else? I think skillset plays a role in why we have a bunch of boxes and grids all over the internet today. Tim Kadlec wrote a great article entitled 'Why RWD Looks Like RWD', in which he says:

"Responsive design is still relatively young. With all the articles and presentations about it, it's easy to forget that. You don't have to look far though to find companies that are just starting to dip their toes into it ... Understandably, people will lean on established patterns to provide a level of comfort as they're working things out. Eventually as people get more comfortable with how to approach multi-device projects, their reliance on these patterns will lessen and they'll start to experiment more."

ESTABLISHED PATTERNS

Today, I would say we are a bit past this point, but at the time of my tweet, we were in the heart of it. Kadlec's article was actually inspired by this tweet by Mark Boulton, a long-time web designer: "I wonder if #RWD looks the way it does because so many projects aren't being run by designers, but by front-end dev teams."

Reading this tweet, which came just a year after mine, made me realise – I had gotten it all wrong. I was blaming the designers, but I think really what Kadlec said was spot on. We are new at responsive web design. There is so much to learn, it makes perfect sense that we would stick with some established patterns as we try to get our feet underneath ourselves.

So what can we do to push ourselves further? How can we break out of the patterns that we've learned and start getting brave with our responsive markup? We can challenge ourselves to explore new patterns when it comes to our frontend markup.

What can we do to push ourselves further? How can we break out of the patterns we've learned and start getting brave with our responsive markup? We can challenge ourselves to explore new patterns when it comes to our frontend markup

With established patterns, we tend to look at a design and break it apart in our minds, working out how we are going to do the markup for it based on our past experiences with similar designs. Sometimes when a design challenge comes along, we get stuck because it doesn't fit inside the patterns we know so well. Push yourself to think outside that box. If you understand the fundamentals of the position property in CSS you can literally do anything. If you don't yet fully understand it, I wrote an article that may help: *netm.ag/positioning-263*.

RWD is the future, no doubt, and some of us are just now getting our feet beneath us. I'm excited to see what we build as we continue to learn and master responsive development. Even more so, I'm excited to see some of you bring some soul to your work, and bring some magic back to the web.

PROFILE

Noah (*@motherfuton*) is a partner at design and development studio Bold. He has a passion for a beautiful web and for teaching the fundamentals of development

SPACE

SPACECRAFT, DATA AND HUMANS

Jesse Kriss casts his eyes to the future of design software for space exploration

In the Human Interfaces Group at the Jet Propulsion Laboratory, we build software for people, just like you probably do. We believe in user-centred design, evidence over opinion, and the importance of learning quickly through sketches and prototypes. Time and budget are always constraints. Sometimes the speed of light is a constraint, too.

You see, some aspects of our work are a little unusual. We're a federally funded lab that exists to accomplish the near-impossible on a regular basis. In short: enterprise software for space exploration.

We communicate with our spacecraft using the Deep Space Network: three sites across the globe with huge radio antennas. As I write this, dish 35 is downlinking data from the Voyager 2 probe. The data, travelling at the speed of light, currently takes 1.24 days to arrive.

Among many tasks for many missions, our group is working on new displays for DSN operators so they can better monitor, debug and correct any issues with data communication. It's critical that they see problems quickly and can immediately jump to the low-level information.

We believe in measured automation. We design software for experts, but we need to make sure they're using their brainpower to do their jobs, rather than for figuring out how to wrangle sub-par software.

The complexity of our missions and of our data are increasing. With Mariner 4, in 1965, you could plot the data by hand (which they actually did, at JPL). The Mars Curiosity rover is currently capturing an enormous amount of data. Mars 2020 will improve upon that again. We must produce software tools that substantially expand human capacity, or we'll hit a ceiling.

When we do our job well, we get to expand the possibilities for science and exploration. Every day, we must focus on ensuring we are building the right tool for the job. We try to avoid guesswork by testing our assumptions and ideas as early as possible, and working with our customers throughout the entire process.

We ask ourselves a lot of questions: Who's our audience? What are we trying to accomplish? Is the thing we think is a problem actually a problem? Does our solution effectively address the problem? Does our design embody our intended solution? Is the design usable by our audience in the context of their work? What's our most dangerous untested assumption?

We can't A/B test our way through design decisions. Sometimes our entire user base is a handful of people working in the building next door. That level of access is great, and it's pretty different from designing a web app for anyone with internet access.

So how do we know what's right? We use a range of techniques to communicate what possible futures might look like. Long before we have detailed interface sketches, we use narratives and storyboards to show our ideas. We use paper prototypes and experience prototypes to evaluate people's use of the tools we have yet to build.

We build software for humans. We design and build iteratively. There's much more software running on the ground than on the spacecraft. It's more straightforward to update ground software than flight software, so we can maintain a certain level of agility in deployment, as well.

At the moment, much of our work runs in a web browser, although we also look beyond the web. We're currently building 3D immersive collaborative software for deciding what Curiosity, our newest Mars rover, does every day, and there are many more opportunities to use augmented and virtual reality for science. These platforms are just beginning to be explored. Our intuition, and design and prototyping techniques, are much less useful in those new realms. We need new tools to build new tools.

PROFILE

Jesse is a senior design lead in the Human Interfaces Group (*hi.jpl.nasa.gov*) at JPL, where he creates next generation interfaces for science and space exploration

AARON GUSTAFSON

Words by Martin Cooper **Photography by** Chloe Wright

The author of *Adaptive Web Design* and ex-WaSP manager chats about his quest to create a mobile web that is accessible to everyone

Interview

INFO

job: Founder and technical lead, Easy Designs
w: *aaron-gustafson.com*
t: @AaronGustafson

"Hiya," a voice says as Skype begins to wake and warm up. The image clears to reveal Aaron Gustafson – bearded, smiling and reclining in his chair. He starts telling his story by explaining his here and now.

"I'm probably best known for being a big advocate of progressive enhancement," he says. "I've been singing its praises since 2004 and that work has become much more relevant in the world of mobile and responsive design … with all these devices with different configurations, capabilities and – in some cases – liabilities. Beyond that, I'm pretty involved with the Open Device Lab."

Rather modestly, Gustafson glosses over his achievements. He's a regular speaker on the web circuit, runs web development consultancy Easy Designs, spearheaded Web Standards Sherpa – a best practices and advice resource for web workers – and has written a book entitled *Adaptive Web Design*.

LATE STARTER

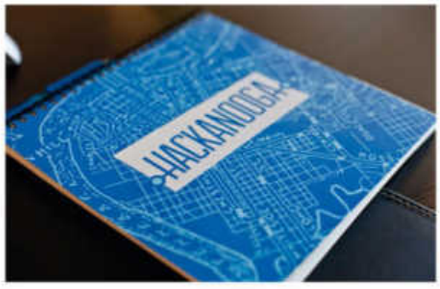
Casting his mind back however, Gustafson admits his love affair with computers was a slow-burner. "I wasn't a prodigy in high school. I had friends who had email accounts but that always seemed too geeky for me. I didn't take any computer science classes," he recalls. "I first went on the web in 1995. I was an avid music fan and the first site I went to was Sony.com. All I saw was this black screen with white words that said 'Image' with square brackets in various positions. I thought: this web thing is bullshit!"

Thankfully, a Jurassic-era Mosaic browser rode to the rescue and Gustafson finally 'saw the web'. "There were lots of table layouts, marble backgrounds with gold lettering and giant Times New Roman text," he laughs.

PAPER FIRST

"I started out doing web stuff in 1996. I was an aspiring journalist and magazine publisher back in college, and I was running a music and entertainment magazine and learned from a friend how to make a web page. I wanted to learn more, so taught myself using a book called *Creating and Enhancing NetScape Web Pages* by Andrew Shafran. It's funny to think back now… It was all hand-coding. I also had a copy of Photoshop 3 on 3 ½ inch floppy discs. That's how I got into web design."

In 1999, Gustafson landed his first paying web job – working for the *Bradenton Herald*



Left Gustafson with his extensive collection of devices **Below** City-wide hackathon 'Hackanooga' is organised by Gustafson's partner, Kelly McCarty, and sponsored by the couple **Bottom** "Work hard, stay humble" is the designer's mantra

newspaper. "Back then," he laughs, "I was the content management system. This was way back before we had CMSes and XML was 'the future'. I went into the *Herald* at 11 o'clock at night and worked until seven in the morning, and I picked which stories went out on the site. I'd pull the stories out of Quark, drop them into Dreamweaver and use Fetch to get them out on our server."

After a few years freelancing and a few full-time jobs, Gustafson ended up working for an advertising company called Cronin & Co. "I came on as their lead web guy – I was the middleman between the design team and the web folks. I had taught myself PHP and MySQL and knew quite a bit about frontend development … HTML, JavaScript and CSS. Coming from a print background, I was able to bridge the gap and make sure the designers weren't designing anything the developers couldn't make and vice versa."

> "The first site I went to was Sony.com. All I saw was this black screen… I thought: this web thing is bullshit!"

A CATHOLIC WEB

Beyond progressive enhancement – a sphere which we can safely say he has made his own – Gustafson a also a prominent member of the web community. He was one of the managers of the Web Standards Project (WaSP), a grassroots group founded in 1988. At that time Netscape and Microsoft split the browser market between themselves and produced platforms that weren't compatible. With designers and developers fighting on two fronts, the web had become fragmented and WaSP – a pack of volunteers and visionaries from around the world – set out to heal it. They advocated open source, consistency, accessibility and portability. As web workers in 2014, we've a lot to thank them for.

So, what attracted Gustafson to the church of web standards? Back in the early 2000s, Gustafson read an article entitled 'CSS Design: Going to Print' by Eric Meyer on A List Apart. Meyer detailed how to create and design print style sheets that would format web content for off-screen reading and printing.

"Back then, I'd become quite the master of table layouts. That was the way we built things, but it always felt weird. So I read Eric's article and thought: there's a lot more to this CSS stuff. I started to read everything I could, and I immediately understood that web standards was a way forward. I saw how fragile the web we were building was … If you had an error in your code and something went wrong, the entire thing could fall apart.

He recalls his time freelancing at Gartner, where there were separate style sheets for each browser, and the team used JavaScript to decide which sheet to serve. "I remember all of the heinous JavaScript we were writing and editing. It was so painful. Then, a light bulb went on in my head – I thought, 'This web standards thing makes a lot of sense!' If we're able to establish standards, it creates a solid foundation upon which we can build better experiences."

From there, by his own admission, Gustafson was like a sponge, absorbing anything and everything he could find about web standards. Writing about the topic was followed by speaking about it, and in 2006 Gustafson joined WaSP himself. He spent his early days working on "some pretty cool stuff", including collaborating with the team at Internet Explorer to improve the JavaScript interpreter and to adopt the W3C's Event Model.

JOINING WASP

Quite soon Gustafson was invited to become a manager at WaSP. "After Glenda Sims, Derek Featherstone and I took over, we worked with Microsoft a bit more to improve Internet Explorer, launched Web Standards Sherpa and began our small business outreach campaign … but eventually we decided it was time to shut down WaSP."

The mission, he says, to a degree, was over too. "The web standards war – trying to get interoperable standards across browsers – was won at that point. There's still work to be done, but we're in a much better place than we were 10 or 15 years ago."

The standards war might be won but Gustafson doesn't seem like a man ready to turn off his Mac and go fishing. Rather, he explains, there are still many risks to the web's apparent wellbeing. "The app mentality is a threat. This idea of creating walled gardens which are 'of the web', but not the web itself. They ▶

Far left A screen print of Erin Crocombe's design for Gustafson's Adaptive Web Design poster contest
Left Another winner from the poster contest, by Guus van Zeeland. "The lines are proposed fold marks to 'adapt' the poster," Gustafson explains
Above Gustafson has written and contributed to a number of web design books

use web technologies and rely on the fundamentals of HTTP, but the resources they provide access to aren't addressable from the web. That scares me. Having indicators about where you can find content is a huge part of the web's power."

INSIDE THE BUBBLE

Another big fear for Gustafson is to do with equal access. "Those of us building websites are technically savvy and tend to have more income to spend, so we have more expensive devices like iPhones and high-end Androids. That leads us to have a very myopic view of what the 'mobile web' is, and what web access on a mobile device needs to be."

To underline his point, he explains he's consulting with a store that's begun selling cheap tablets. "I asked the web team if they're testing on the devices they sell. There was stunned silence on the phone. We're surrounded by high-end devices and get into the mindset that this is the mobile web. We miss the low-end devices with Android, a bad processor and a crappy screen."

This brings us to Gustafson's primary hunting ground: adaptive design. "Progressive enhancement underpins everything I do," Gustafson says. "The entire idea is that you're creating sites, content, web pages, whatever it is, without placing any technological restrictions on the users. With progressive enhancement, you focus on the content and the key tasks of the page, and build up the experience from there, layering on different technologies. The experience is more of a continuum from a basic one that might just be text with links, right up to a fully interactive one."

Gustafson's philosophy centres around giving different people – or more correctly, different devices – different experiences. "It's all about recognising that it's okay for people to have different experiences of an interface as long as those experiences are positive and as long as they can accomplish the tasks they set out to," he says.

ADAPTIVE VS RESPONSIVE

"The name of my book, *Adaptive Web Design*, came about because 'Progressive Enhancement' has a very sterile sound to it. We liked the idea of a web experience that could adapt to the user." Little did Gustafson know, the book version of Ethan Marcotte's 'Responsive Web Design' article would be launching at around the same time, prompting a flurry of 'adaptive versus responsive' discussions within the industry.

> "We're surrounded by high-end devices and get into the mindset that this is the mobile web"

"Ultimately the two approaches are very much aligned," says Gustafson. "According to Ethan, responsive design is three things: media queries, fluid grids and flexible images. Those three things absolutely should be part of a page's progressive enhancement. That said, you can build a desktop-first responsive site and work from the largest size down to the smallest. The lowest common denominator devices don't have media query support... So, if you were to flip it the other way and practice responsive web design from a mobile-first perspective, that aligns perfectly with progressive enhancement. For me, responsive is a technique that comes under the umbrella of progressive enhancement."

Gustafson is at pains to point out that building with progressive enhancement doesn't necessarily mean sites cost more to make. "When you get into this mindset of progressive enhancement, it adds very little. You start to realise how browsers work. An underlying feature in HTML and CSS is fault tolerance ... The system can cope with problems without throwing errors to the user."

In HTML and CSS, he points out, browsers ignore what they don't understand. Recognising this is key to writing HTML5 such that browsers that don't even understand HTML4 will still render the content.

"The browser will expose any text inside the element, it'll just ignore the element," he explains. "So, you have the brilliant system of fallbacks within HTML and within CSS that empower you to drive a baseline of support for older devices whilst, at the same time, being able to reach for the stars."

generate

The conference for web designers

SAN FRANCISCO 15 JULY 2016

Learn cutting edge CSS, JavaScript, user experience and web performance techniques, and much more!

MIKE MONTEIRO
DESIGNER, ARTIST
www.mikemonteiro.com

SARAH DRASNER
SENIOR UX ENGINEER
www.trulia.com

PETE SMART
DESIGNER, SPEAKER, WRITER
www.petesmart.co.uk

LYZA DANGER GARDNER
DEVELOPER
www.lyza.com

FOLLOW @NETMAG FOR MORE DETAILS OR VISIT:

netm.ag/1lr9LWq


6°c
BBC RADIO 1

INNOVATION

DESIGNING FOR DISPLAYS THAT DON'T EXIST YET

Illustration by Ben Mounsey

Devices are changing all the time. But what if the device you were designing for wasn't flat? Or rigid? What if it hadn't even been invented yet? **Rosie Campbell** did just that …

What if you could decorate the walls of your home with electronic wallpaper, and change the content at will? This might sound like science fiction, but with a trend towards larger, more flexible, lower-power displays, we believe some form of 'smart wallpaper' could plausibly be with us in the not-too-distant future.

Smart wallpaper is just one example of how our screens might evolve. Not long ago, almost all our web browsing took place on a desktop monitor. Now, many of us own numerous electronic rectangles of all shapes and sizes, and we're increasingly adopting novel technology like wearable tech, flexible displays, virtual reality headsets and more.

As developers and designers, one of the challenges we face is how to create compelling content for such a variety of unconventional devices. How can we anticipate how they will be used? And how can we create beautiful, intuitive content on such unfamiliar platforms? Additionally, at the BBC we need to understand how we can use these devices to enhance media experiences for our audience.

The fact that these devices are so experimental (some of them don't even exist yet) can make creating new experiences for them seem like a very daunting task, but it is the only way we can test ideas and find out what does and doesn't work. We believe that by anticipating emerging technologies and creating prototypes for user testing, we can refine our ideas so when they do eventually get manufactured, we will be ready for it.

Recently, we built a browser-based smart wallpaper prototype. I'd like to share some insights we gathered during the project, in the hope that it might help guide other developers and designers facing similarly unconventional tasks.

LET UX DRIVE DESIGN

The idea is to let the user experience, rather than the technology itself, drive the design. This concept is sometimes known as 'design fiction': taking inspiration from science fiction to create real-world prototypes in order to envision future technology and test out ideas. Think big about the experience you want to create for the user, and then let the technology catch up with you.

This is pretty much the opposite of the approach traditionally taken: invest a lot of time and money inventing cool new technology, leaving the intended user experience as an afterthought. Inevitably, the result is underwhelming, the use cases feel contrived and the product struggles to make an impact.

Instead, if you can create a sense of excitement and anticipation for a technology through how it could be used, the market appetite could drive manufacturers into developing the technology sooner. And if you can somehow prototype your ideas and obtain user feedback, you could provide insights that ensure the technology will genuinely suit its audience.

In practice, this means having an 'ideas amnesty': an open discussion to tease out all of your thinking (no

matter how ridiculous!) about how the technology could be used. Try not to censor yourself – no idea is too silly – you're trying to get people enthused. The next stage is to choose the most plausible and engaging ideas to take through to prototype.

CONSTRAINTS BOOST CREATIVITY

The problem with futuristic technology is that, often, you won't know exactly how it will evolve. This requires us to make some assumptions. With smart wallpaper, we thought it unlikely to be touchscreen, because it would be irritating having to get up to interact with it. Instead, we posited that interaction would be gesture-based via a mobile device. We also assumed the display would be low power to save electricity, and non-backlit so as to be unobtrusive.

We came to the conclusion that it might be less like an LED monitor, and more akin to e-ink. This obviously presents challenges, as e-ink is not optimised for video: it has low-refresh rates, low power and no backlighting. But in considering these limitations, we actually found they helped us think more creatively, and we ended up with more novel use cases.

Rather than simply imagining a giant wall of video, we began to think about the wall as a companion device to the video-optimised TV, adding stats and data to complement live sports and music events, or using panoramic images to create a more immersive experience. To test interaction methods, we even built an immersive 'Hide and seek' game for children!

Of course, it is also important to be flexible. Our assumptions may turn out to be incorrect, and if so we don't want all our work to go to waste. That's why our next insight is so important …

STAY HARDWARE-AGNOSTIC

By nature, there are lots of unknowns when dealing with experimental technology. Because of this, it's risky to rely on a specific architecture in case the technology happens to evolve differently. We found the best way to avoid this was to use standard web technologies.

Since we can be pretty sure any new devices will be internet-connected, our work should be pretty future-proof. We wrote our 'Smart Wallpaper' application using HTML5 Canvas and JavaScript, with a RESTful API and Node.js backend. In our prototype, we simply used giant projector screens displaying fullscreen browsers to simulate a smart room.

Another advantage to using standard web technologies is that anything with a browser can represent a wall, so we don't have to wait for full-blown smart wallpaper to exist for our work to apply. As people get larger screens, and more of them, our system can start treating these as walls and delivering new experiences.

PUSH RESPONSIVE DESIGN

We expect our web content to look beautiful and feel intuitive, no matter what the device. This links to the idea of responsive design, which aims to adapt content for different screen sizes. But as the variety of devices, displays and screens increase, we need to start expanding our understanding of responsive design. We can no longer assume our screens are going to be rectangular, flat or have other properties we've come to expect.

With smart wallpaper for example, we have to contend with furniture obscuring the display. Additionally, we have to find the right balance between automating the design and allowing the user to have some control over layout. We also have to determine which type of content suits which device.

As the variety of devices and displays increase, we need to start expanding our understanding of responsive design … We can no longer assume our screens are going to be flat, rectangular, or have other properties we've come to expect

All this presents a new challenge for responsive design. We need to go beyond arranging rectangular elements in a rectangular space, and start looking for solutions elsewhere. Perhaps we can use computer science algorithms like bin packing, or data visualisation principles such as clustering? Maybe we can take inspiration from the natural world? Can we use JavaScript to automate these new responsive layout systems? To what extent should they be automated?

These are some of the questions we would like to answer, and we imagine anyone hoping to create novel experience for emerging technology will face similar ones. By thinking about these problems now, we might just bring the future that little bit closer.

PROFILE

Rosie is a research technologist at BBC R&D (*bbc.co.uk/rd*), and has been working to prototype new content experiences for emerging broadcast platforms

net

PRINT & DIGITAL EDITION

TRY OUR DIGITAL MAG FOR FREE!

Discover the no.1 choice for web designers and developers. Each issue is packed with the latest trends, technologies and techniques, plus exclusive video tutorials. **Don't miss it!**

PRINT: myfavouritemagazines.co.uk/NETMAG15

IPAD: netm.ag/net-ipad-uk OR netm.ag/net-ipad-us

Available on amazonkindle · zinio™ · Available on the App Store · nook by Barnes & Noble · GET IT ON Google play

SCOTT JEHL

Words by Martin Cooper **Photography by** Chandler Williams

One of the faces behind a landmark project in RWD reveals how to prioritise performance, and why we need to start being more responsible

INFO

job: Full-stack developer, Filament Group
w: *scottjehl.com*
t: @scottjehl

The 2011 Boston Globe website project (*bostonglobe.com*) – which saw design and frontend development studio Filament Group team up with Ethan Marcotte – is seen as a watershed moment in the evolution of responsive web design. It was one of the earliest large-scale implementations of responsive design, and served as a refining ground for the approach and the processes that underpin it. One of the key figures on the project was Scott Jehl, a web designer at Filament Group, who has also worked with clients including Lego, Global News and eBay.

This game-changing moment for the web design industry was closely followed by a life-changing event for Jehl himself. He and his wife hopped on a plane to Southeast Asia, where they spent eight or so months travelling around. "I left after I had finished up the Boston Globe project and, taking off [for Asia], I felt like I was in a good spot. I felt like I had a lot of answers," he recalls.

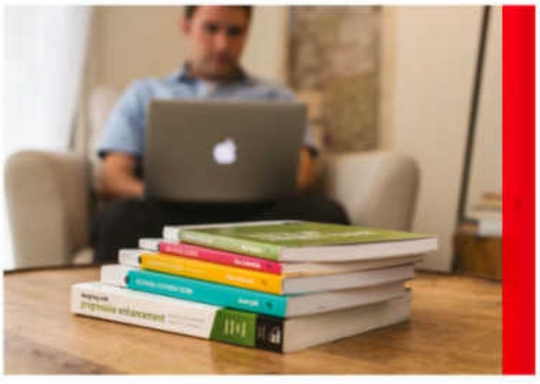
On the road, Jehl worked remotely for the Filament Group while his wife volunteered at a hospital. The couple put down roots for a few months, and Jehl found himself needing to access the internet using the local services. This came as something of a shock.

A DIFFERENT REALITY

"[In the US] we have it pretty cushy," he says. Wi-Fi, 4G and Apple's latest gadgets are all taken for granted. By comparison, accessing the web via mobile networks in SE Asia – as many of the locals do – posed more of a challenge. It relied on a combination of basic protocols, luck and the data showing real determination to reach its destination. "I really got to feel what it was like living in someone else's shoes," Jehl adds.

Experiencing this divide between the digital haves and have-nots brought about a new determination in Jehl. Having left the US filled with confidence won from the Boston Globe's success, Jehl returned humbled and appreciative of a larger truth: design shouldn't just be responsive, it needs to be responsible too. "I realised that there are a lot of big problems still to be solved," he reflects. "The focus for me became performance, particularly on slow and intermittent connections."

The realisation was so strong it prompted Jehl to pen a book on the topic: *Responsible Responsive Design* (*netm.ag/responsible-267*),



published in late 2014 by A Book Apart. The title explores ways to develop for a truly global audience – regardless of network capabilities – and it is already making waves in the web community.

RESPONSIBLE DESIGN

Building sites that are crafted to work on slow connections may sound like the morally right thing to do, but there's also the business side to consider. Why should a US client with a domestic customer base spare their time and money making sure they have a site that will work in the developing world?

"What you have to remember is, even in the US, 3G is still the most common connection speed," Jehl replies. "LTE hasn't dominated – even in cities. I go into Boston and there are blackspots."

As for the business case, designing sites and applying optimisations that make the web feel quicker is a pretty easy sell: "If a person is browsing with a prepaid SIM card, every byte they download has a real cost. And most sites on the internet are a megabyte-and-a-half or more in size. That's pretty heavy."

So, designing sites with performance in mind helps everyone. However, Jehl admits that technically, it can be a tough proposition. For a site to feel slick it needs to render in one second or less on Wi-Fi, and in around two seconds over 3G. The perceived wisdom about how you maximise performance is constantly evolving. Jehl recalls the early days of responsive, when there was an immediate focus on the size of the site. We were building sites that addressed a lot of different usage scenarios at once and this, inevitably, made those sites heavier.

"Fluidity was a main concern and so was performance," he elaborates. "One of the areas people directed their attention to was the raw file size. How big was the site? How many bytes were being downloaded?"

Lately there's been a shift. The technical focus has moved on to exploring how we can deliver lighter sites, while still maintaining the same rich experience. One technique, Jehl says, is prioritising what arrives first. By managing the order in which content is displayed on screen, we can greatly influence how the performance of the site is perceived. Although, of course, changing the order in which content arrives doesn't reduce the weight of the site itself.

> "3G is still the most common connection speed in the US. LTE hasn't dominated, even in cities"

PERFORMANCE ISSUES

As we talk, it becomes apparent that Jehl believes performance should be the star that guides a project team. Performance should be considered at every turn, because it has a profound effect on a site's ability to reach an audience. "If a site is not delivered quickly, you can't begin to think about usability," Jehl says, simply.

"Our client interactions always start by getting everybody on the same page," he continues, explaining how to get a project off on the right foot. "Performance becomes baked into everything we produce throughout the course of the project." Again, Jehl makes the point that convincing clients of the need to focus on performance is never a difficult proposition. A highly performant site will naturally present low access barriers, which means more paying customers will be able to access it.

TACKLING BUDGETS

We move on to discuss performance budgets. Jehl explains there are, in his experience, two common approaches. There's the overall budget of how much code you want to deliver, and there's the budget of how much time you can afford to spend rendering this code on various connection speeds. Neither metric is perfect.

At Filament Group the focus usually rests on time-based metrics. The team's budgets are incorporated into their build tools so, when they make changes to a codebase, the code is optimised and then tested in a browser. This testing reveals whether the changes to the code have made any performance regressions against the budget – is it still loading within one second on Wi-Fi, and under two-and-a-half seconds on 3G?

"If we regress," Jehl says, "we have to go back and figure out what happened. We tend ▶

to approach the budget that way, but I've seen other companies have very successful budgets that are based on file sizes. At Etsy – I don't know if it's an unwritten policy – but when somebody adds something heavy to a page, they have to remove something of the same weight. That works too."

EXPLORING INLINING

Finally we move on to exploring how a team can boost performance after they've finished developing and editing their code. "There are lots of automated processes we can use," Jehl reveals. "We can let our tools combine files so we make fewer requests over the network. And we can start to deploy inlining." Inlining involves serving up some of the CSS needed for a page in the same file as the markup. With the HTML and CSS combined, the number of server requests needed to build a page are reduced.

Inlining is, in Jehl's opinion, simply incredible. So incredible, Google commonly deploys the technique, which is one reason why its sites are so highly performant.

"Inlining's a comparatively new thing that's come along in the last year or so, and it's really taken off," Jehl enthuses. "It's great because it offers such a dramatic improvement. It can, in some circumstances, cut load times in half. That's because a site can take two, three or more round trips to a server before it can start to render something usable."

You shouldn't, of course, develop code with CSS and HTML combined. It's simply too messy, and, as Jehl points out, blending your concerns is bad practice. "On the flip-side," he counters, "if you're making the simplest one-page website, it's common – and actually smart – to put a style element in the head of a document, and add your CSS there."

It's smart because when the browser encounters externally linked CSS it will effectively stop everything it's doing. It will then request, download and parse the CSS. When all that's done, it'll get back to the business of rendering your page.

> "Inlining offers such a dramatic improvement. It can, in some cases, cut load times in half"

"So, yes, develop your code in silos," Jehl sums up, "but when it comes to optimising your code, it's a very different concern. Inlining is something we want to have our tools automate after we've finished developing. It's a production step."

TOMORROW'S TECHNIQUES

Jehl's relentless quest for better performance doesn't end with inlining – he's currently eyeing up the possibilities of HTTP/2. The problem with HTTP/1.1 is that, because it only allows one request per TCP connection, loading multiple assets efficiently is difficult. To get round this limitation, browsers can run multiple TCP requests in parallel. However, this is a limited fix, because too many TCP requests can lead to network congestion and resource conflicts. Multiple requests also produce a lot of data duplication which, again, harms performance.

What's attractive about HTTP/2, Jehl says, is that it makes a lot of today's performance hacks unnecessary. Certain browsers (Chrome and Firefox, for example) now support HTTP/2, but it's still not widely accessible. "The problem is, servers need to be converted so they can communicate in this language," Jehl explains. "So it's not something that's going to happen overnight."

HTTP/2 also offers new features. The server can, for example, push assets to the browser in the same round trip that another request is made. "The browser can say 'please send me this HTML document' and the server will think: 'In addition to that HTML document, you're going to need this CSS file'," explains Jehl, excitedly. "The server will send both back to the browser in one request. So HTTP/2mimics that same inlining workaround we're using."

With that, Jehl draws our chat to a close. Rattling though his complicated schedule, he explains that he's off to Australia in a few days. It comes as no surprise to find he'll be talking about performance.

PERFORMANCE

HTTP/2

Michael Gooding on why the latest version of HTTP will be a game-changer

HTTP/2 has arrived and its goals are simple: to improve the performance of HTTP by targeting the way the protocol is used today. In other words, to load websites even more quickly. The new standard has a host of features that address today's web usage patterns. I'll explore them here.

MULTIPLEXING

Multiplexing is a way of requesting and receiving more than one web element at a time, and it is the cure for the head of line blocking in HTTP/1.1. Currently, each client request needs to wait until the server's response to the previous request arrives. If a request stalls, the whole page download is delayed.

HTTP/2 is a binary framed protocol that means requests and responses are broken up into chunks called frames, with meta information that identifies the request/response they are associated with. This allows objects to overlap on the same connection without causing confusion, and for them to be sent/received in any order. A first request may take longer to complete but it won't hold up the delivery of any subsequent objects, meaning faster page load.

HEADER COMPRESSION

Headers have grown in size. Using a combination of lookup tables and Huffman encoding, HTTP/2 can reduce the number of bytes sent in a request down to zero. In an average web session, compression rates above 90 per cent are not uncommon.

With HTTP/1.1, a modest page with 75 objects and an average header size of 500 bytes might take the browser four TCP round trips just to request the objects. The new protocol doesn't improve the size of responses, but on the request side the results are significant. With the same parameters and 90 per cent compression with HTTP/2, a browser can send all the requests in a single round trip.

DEPENDENCIES AND PRIORITISATION

Browsers have introduced pre-loaders to ensure they request the most important stuff first. If, in the new model, a browser simply requests everything at the same time and allows the server to return objects as quickly as possible, there will ironically be a reduction in page performance, because key objects required for rendering may not be delivered first. The designers of HTTP/2 have addressed this: by communicating to the server what objects are dependent on other objects and listing the priorities, the server can make certain the critical data is delivered to the browser right away.

SERVER PUSH

Server Push allows the server to send the browser an object before it is asked for, providing instant page delivery even in the worst conditions. However, in order to push the correct objects without wasting valuable bandwidth, the server needs to know what the user is likely to need next, and what the state of the browser cache is. Whilst HTTP/2 provides the tool for Server Push today, I am sure we will see some innovative uses over the coming years.

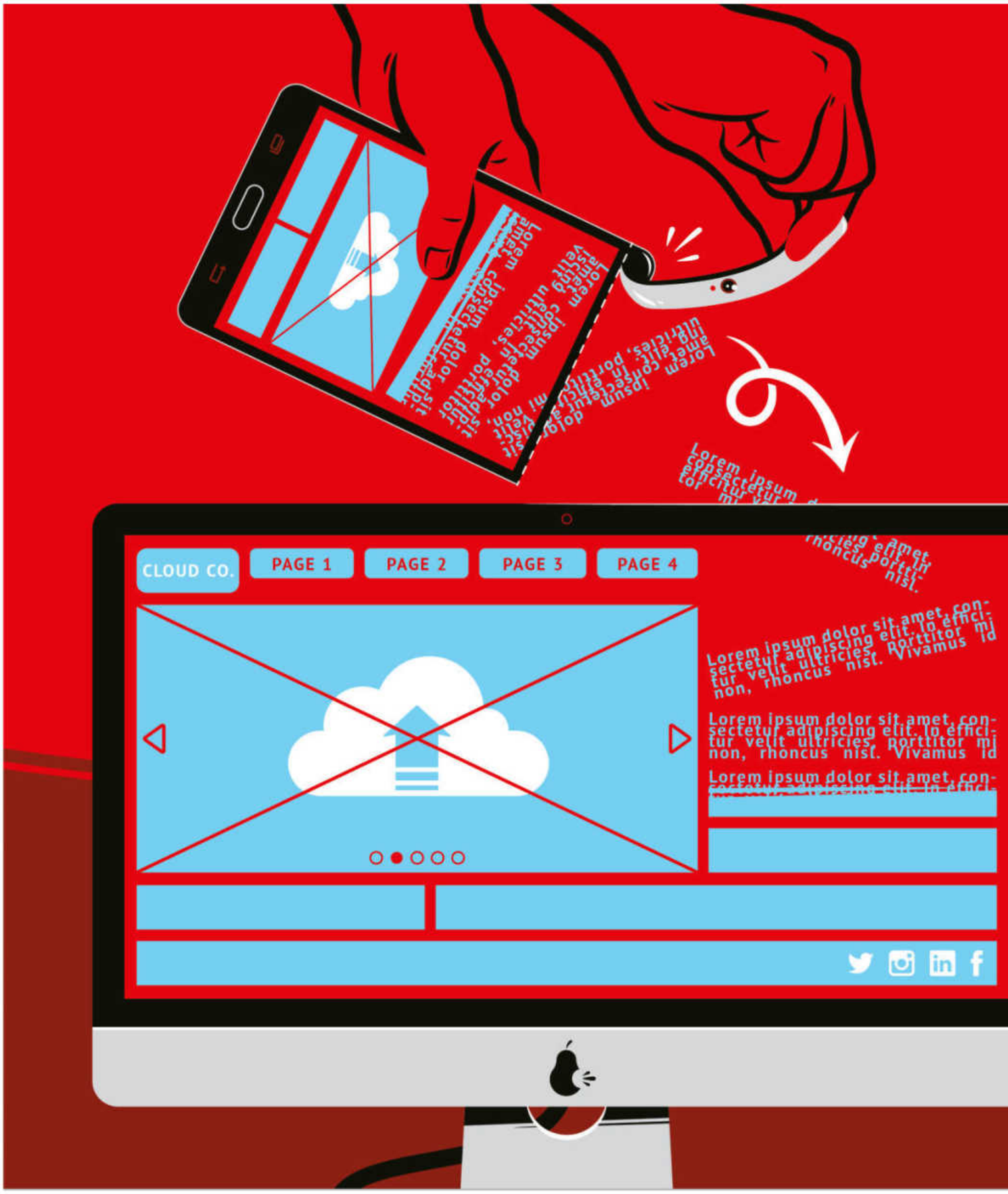
... AND FOR THE EVERYDAY USER?

No one will have to change their website or applications to ensure they continue to work properly. HTTP/2 is compatible with existing code and APIs, with the one major difference being you will need to deliver using TLS if you're not already. Organisations should, however, start thinking about how they can capitalise on the new performance and security capabilities.

Read the extended version at netm.ag/http2-271

PROFILE

Former developer Michael (@Michael_G_81) works at Akamai to identify performance bottlenecks and solve complex problems for some of the biggest websites in the world


CLOUD CO.
PAGE 1
PAGE 2
PAGE 3
PAGE 4
Lorem ipsum dolor sit amet, consectetur adipiscing elit. In efficitur, velit ultricies, porttitor mi non, rhoncus nisl. Vivamus id
Lorem ipsum dolor sit amet, consectetur adipiscing elit. In efficitur, velit ultricies, porttitor mi non, rhoncus nisl. Vivamus id

PROCESS

RESPONSIVE CONTENT MODELLING

Illustration by Ben Mounsey

Steve Fisher takes a look at how to nail down the core piece of content around which your whole site orbits

Without content, a website is a black hole. Finding that core piece of content from which all the rest of the pages and elements hang saves us from being sucked into the black hole of building fancy buckets to hold Lorem Ipsum. Finding the core content of your website means the entire team (vendors, stakeholders, audiences) will walk away with a common understanding and vision. Every time you write or revise, you'll think about that core piece of content and refocus. The content authoring experience will become what it was always meant to be: dreamy.

To find this core content, we use something called a content modelling workshop. In this article, I'll take a closer look at our process.

TEAM TALK

First off, you need people in the room who have the right passion and information. A team might include a UX lead, content strategist, developer, client project sponsor, client IT lead and client content specialist. You may not have all these specialists for every project, but there are probably people with these general responsibilities in their job descriptions. The goal is to involve people who understand the problem, are passionate about solving it, and have the authority to make decisions.

Responsive content modelling is a team-building exercise, but with fewer trust falls and more Post-It notes. Trust is crucial; everyone needs to feel safe to speak freely. For this, getting everyone together is key. This means being locked in a room together for anything from a couple of days to the better part of a week. Take breaks as a group, and get lunch delivered.

BE PREPARED

In preparation for your workshop, you need to inventory everything you currently have. You have to know what exists to know its purpose and to see patterns.

Next you need to set some high-level goals for the site. So you don't fall into the trap of letting opinions guide your decisions, it's important you know your audience and its needs. If you don't know who you're talking to, it's pretty damn hard to write for them.

You then need to work out the UX vision. Fire off a quick draft to get it out there and ensure it is pointing the group towards the right target. Finally, establish the project's design principles. These are the guideposts for all decision-making during the project. They're the values or 'why' statements that will keep you moving towards the vision. Think of them as the guiding principles of the project.

Nailing down the 'who, why and what' is crucial. You need to know your purpose because you can't know what your central content type is without knowing who you're communicating with and why.

CONTENT TEMPLATES

With this framework established, you can dig into the content modelling phase of your workshop. You're going ▶

to prioritise every content type. Here you look at all the content views you'll need: the homepage, landing page, standard content page, news releases, application templates, advanced search and so on. Often there will be an established list, based on an earlier contract.

You want to begin to prioritise all these content views by comparing content templates with the vision and goals. You need to consider questions like: Where will the content live? Where will the most important content to the primary audiences be found? Will it live on the homepage or a landing page? Is this a page where search is dominant?

In this phase, you must agree on everything; no compromise. This is a must. Focus on audience needs and toss opinions aside. This is the secret to delighting audiences and giving them what they need. It will be important to know how to argue well and speak your mind, so you need a strong facilitator. This can be anyone who understands a project's process and purpose. The best facilitators are often the experience architect or the content strategist.

Remember, there is no such thing as the client team and the vendor team: there is only the project team. You can't successfully find that thing that bonds the team and the content by working separately.

LOOKING FOR THE CORE

Go you! You found and prioritised the core content templates. Now you need to get more granular and find the unique content type everything orbits around. To do this, you list the discrete content types of your content templates: page title, contact module, featured image, teaser copy, main body copy, related items, and so on. At this point, this may include items that you're not sure you want included. We'll sort these out later.

Attach a priority level of one, two, or three to each content type or item on the list:

- **1 – Essential:** This view wouldn't be able to function or communicate the core message without this item
- **2 – Great to have:** These items fully support the core purpose of this view
- **3 – Nice to have:** If these items didn't exist it wouldn't have a huge impact on the view

If you start to label items with a four, they're probably pointless. Everything should point people towards the core purpose of this content. Some priority three elements might reveal themselves as necessary and some priority ones could get demoted. Things will shift as the team progresses.

NARROWING DOWN

You're not done yet! Now you go back through the three priority groups and assign a priority order to each element within each group. Be prepared to duke it out, laugh, cry, and sweat through this. It's not about compromise, it's about focusing on what's best for the audience. This is why you established a framework through our UX vision, design principles and high-level goals. Return to those guideposts for every decision.

Something must be the very first priority and something must be the last. Take it seriously and resist the urge to quit – I promise it will be worth it. This is a less technical, but more human-centred method of content modelling. When you're done, every content type will have a unique priority, and content type 1–1 is the most important. This is your core content type.

In a multi-device world, content should adapt to all platforms and be represented consistently. You can do this by making sure that your core content type is the first thing users see on any device

PRIORITY 1–1

In a multi-device world where content can live anywhere, this is one of the most important things you can do to make your content successful. Content should adapt to all platforms. You can do this by making sure that 1–1 content is the first thing users see on any device.

Regardless of content being accessed on a Pebble watch or a billboard, the key message remains clear. And for the 90 per cent of people who juggle between devices, they're seeing the congruence of the content.

Finding the core piece of content really is the key to every web project, big or small. It will make your projects more successful and help you build a better, more thoughtful web, one content type at a time.

PROFILE

Steve is the founder at The Republic of Quality (*republicofquality.com*) and the co-founder of The Design & Content Conference (*designcontentconf.com*)

SUBSCRIBE TO COMPUTER ARTS

Get 13 issues of the industry-leading magazine for graphic designers in print and on digital devices for just £66

SAVE UP TO 54%

Enjoy the tactile beauty of our print edition and the portability of our iOS and Android editions in one package. A subscription to Computer Arts gives you behind-the-scenes access to top studios, analysis of the latest trends, and inspiration that lasts all year.

Subscribe online today:

www.myfavouritemagazines.co.uk/camag16

T&Cs: Prices and savings quoted compared to buying full priced print and digital issues. You'll receive 13 issues in a year. If you are dissatisfied in any way you can call us to cancel your subscription at any time, and we'll refund you for all un-mailed issues. Full details of the Direct Debit guarantee available upon request. Prices correct at point of print and subject to change. Full terms and conditions: myfavm.ag/magterms. Offer ends 31/12/2016.

INFO
job: Web designer
w: *bradfrost.com*
t: @brad_frost

BRAD FROST

Words by Martin Cooper **Photography by** Noah Purdy

The web designer on cutting out showmanship, turning traditional publishing on its head, and why the big agency model is shifting


Enjoy
HOT SOUP

Brad Frost is not a man for fancy job descriptions. "I'm a web designer, I leave it simply like that," he smiles. "I don't like getting too far into the weeds with titles." These days he specialises in helping clients move to a responsive way of working. This, he's keen to point out, isn't just about making web pages jump, morph and fold as the viewport changes size. That's the easy bit. Any web designer worth their salt, he reckons, can knock a responsive site together in a few days.

The difficult bit is what goes on behind the scenes. As such, these days Frost is more involved in helping companies adapt their workflows, sales, processes and structure so they can work responsively. Add that to his writing and speaking commitments, and he's a busy man. "I keep my dance card pretty full," he laughs.

FIRST DATES

Frost has established an impressive roster of big-name clients, including TechCrunch, Entertainment Weekly and Mastercard, to name a few. Happily, the industry has now got to the point where, at that all-important first meeting, Frost doesn't feel the need to explain responsive design any more. Many of his clients have already dipped their toes into the water and embraced the technology.

"The traditional model of contractors and agencies working with clients is an awkward dance," Frost says of the mechanics of that first meeting. "It's about impressing people and wowing them into hiring you. That's not how I think it should be."

Frost is emphatic that if you're a web design shop, this meeting should "go deeper than splashy pitches and showreels", and instead focus on having open and honest discussions.. It should be about facts, and taking the time to have an honest discussion.

Sadly though, he says, the desire to impress the client can sometimes get in the way of more important conversations. Those conversations should include, he says, discussing where the client feels they are now, and where they want to be.

So what kinds of things does Frost look to uncover? "Very often you have an idea about their goals," he says. "Maybe the client has a six-year-old website that's crusty and hard to maintain and, increasingly, they're seeing more numbers coming from mobile."

In such situations, having an objective outsider present can help bring clarity to the situation. "Those can be difficult conversations to have," Frost explains, "and I feel like as an outsider I'm sometimes able to play the role of therapist, to get people to open up."

NAILING DELIVERABLES

At this early stage, there also needs to be a discussion about deliverables. "You really need to set people's expectations about what it means to build a site in 2015," Frost says. "And that means talking through processes."

One of the biggest challenges he faces is helping people understand that it's OK to proceed without a hi-fidelity image of what the final site will look like. That leads to unrealistic expectations because, at the end of the day, a picture of a website isn't a website.

Frost laughs: "Stephen Hay has a great line where he says, when you show a client a full comp, what you're actually saying is: 'Here's a picture of what your website will never look like!' And it's entirely true."

Once more, Frost returns to the temptation, when working with big agencies, to add an element of showmanship during these early meetings. "I've seen people paint these dramatic, blue-sky comps and put them in front of a client with a grand Don Draper reveal. It's like: 'Wow! Aren't you impressed?' Sure, they might be impressed. But, at the end of the day, it's dishonest to design things in that way. It's not a smart idea to produce high-fidelity comps too early in the process."

> "I feel like as an outsider I'm sometimes able to play the role of therapist, to get people to open up"

The reason being that the web is fluid. It has 100,000 viewport sizes and static comps don't give a holistic impression of any proposed site. "This is something I've been talking about for a long time," Frost continues. "Historically, we've equated design with aesthetics. Sure that's important, obviously. Colour, typography, things like that – I'm not discounting them. But often, people paint a very myopic picture of what the design actually is."

What's more, a static comp fails to represent critical considerations like performance and ergonomics – which have a huge impact on how users perceive a client's brand.

MEETING CLIENT 2.0

The success – or failure – of the sign-off process, Frost maintains, can be traced back to those early client conversations. At that stage it's good practice to set expectations about how you intend to present incremental work. The key is to agree on an iterative, as opposed to a waterfall, approach.

The process, Frost says, "shouldn't be based on call and response – 'we present something to you, you react to it'. It's meant to be a conversation."

AGILE ASPIRATION

Surely though, clients who are fully conversant with the subtleties of responsive design are rare. Aren't more traditional art directors hard-wired to ask for comps to sign off? According to Frost, these days clients tend to aspire to a more iterative, collaborative and agile way of working.

The reason is, partly, an economic one: it's simply a waste of time burning huge numbers of deliverables. However, there are certain pressures on agencies that can mean they don't always deliver that in practice.

"I'm working with a big client right now whose history is pretty entrenched in their catalogue," Frost admits. "So, historically, their process has very much been: 'Here's a picture of the website, sign off on it, Mr Main Creative Director'."

"Historically, big agencies have felt a lot of pressure to 'prove' themselves to clients, which can lead to those grandiose pitches and blue-sky design explorations," he ▶

continues. "But in order to truly address clients' needs, both clients and agencies need to evolve their processes to work more collaboratively and efficiently."

CALL FOR UNICORNS

So, what's Frost idea of a perfect team and structure? "In an ideal world everyone is a unicorn," he quips. "They magically traverse between Ruby, Photoshop and PHP." In the real world, Frost says he's seeing a shift toward small teams that are doing incredibly good work.

"You have people like Paravel out of Austin, Cloud Four from Portland, yiibu from Scotland and Dan Mall's SuperFriendly in Philadelphia," Frost explains. "That's why I'm curious about the monolithic agency model. It doesn't require 1,000 people to make a 1,000 page website."

All you need, he says, is time and a small group of cross-disciplinary, 'T-shaped' crafts-people. And to make best use of them within a big organisation, Frost advises, you need to set them free on a pilot project.

"Let them – your A-Team, your Navy SEALs – set the pace. Don't shackle them," he says. "That's the most effective, most realistic way to get an entire organisation on-board. Just let them do their stuff. See if it works – and when it does, roll that success out across the whole organisation."

From a strategic point of view, Frost says the roll-out should be done by the client's own people. This approach ensures everyone is on-board with the changes, and helps people feel invested in the new site.

ITERATION

There's another trend that's shaking up the way websites are being managed. Firms such as Airbnb, Amazon and Etsy are embracing small, data-driven, iterative changes. They may never again need redesigning because they're being constantly improved and refined. In Frost's opinion, the days of the blanket redesign may well be a thing of the past.

> "It's stupid to spend a million dollars on a redesign when you could be making subtle changes and seeing benefits"

"Back to the big agency model: that's their bread and butter. 'So, you've got a crappy old Flash site? HTML5 is all the rage'. And, in three years' time they come back and do it again."

He explains that some of the firms adopting a more iterative model are seeing huge commercial gains coming from small improvements. "I'm not saying companies shouldn't rebrand and move forward," he says. "People are just realising it's stupid to spend a million dollars on a redesign when they could be making subtle changes and seeing benefits."

Of course, there are times when a full-blown redesign is the only option. To help make things more straightforward, Frost suggests thinking strategically and embracing pattern libraries. This way, the process will be much less painful because there's already a good system in place. "By embracing patterns, you're setting the stage for future redesigns," he says.

TAKING ON PUBLISHING

Finally, we touch on the other project taking up Frost's time. He is writing a book – and it's a unique one. Called *Atomic Design*, it focuses on how to craft effective interface design systems (*netm.ag/atomic-265*).

What's really special is this particular book is being written live on his site, with the code hosted on GitHub. "I'm trying to be as open as possible with the book's creation," he explains. "I have a mailing list and I'm writing about the progress and process, amongst other things."

This means that the community can comment on the work as its being made. More importantly, the readers can learn from the book straight away. It seems publishing – like that big agency model – is facing some exciting changes, and becoming more flexible to meet the tastes and needs of modern audiences.

Q&A

TIMONI WEST

The Foursquare and Flickr veteran on how design can influence user habits, data sharing and privacy

INFO

job: Design head, Unity's Labs team
w: *blog.timoni.org*
t: @timoni

Why don't you introduce yourself?
TW: My name is Timoni West, and I head up design on Unity's Labs team, where we're working on VR development and creation tools. Over the years I've worked with over 30 different tech startups and companies, doing all kinds of design.

Where does UX end and UI design begin?
TW: I do both myself, and I find it awkward to draw a line in the sand where you pass off unfinished ideas for someone else to refine or reskin. Additionally, so much of design now includes animating behaviours … so where does that live? Theoretically, it's the heart of user experience, but in reality, most people don't animate wireframes.

You've spent a lot of time making interfaces that encourage people to put data online, like Foursquare, Flickr and Scribd. What's the secret?
TW: I'm glad you asked! In order to create a social network that makes you feel good, you need three things: a way to easily make stuff, a way to easily see and react to other people's stuff, and a way to see how people have reacted to your stuff.

Social networks like Myspace or Friendster had these components, sort of. But then better social networks came along that made it easier and faster to make, respond, and see feedback, and so everyone migrated. Facebook, Twitter, and Instagram are obviously killing it on all three points.

Will designers feel pressured into making interfaces that dupe people into sharing too much?
TW: Only if their company found a profit model around duping people into sharing, I suppose. I just installed Secret recently and people are certainly sharing things they wouldn't share otherwise, but they're not being duped. The real concern is how secure Secret's codebase is.

Is responsive design really just about the size of the screen? What about the internet bandwidth?
TW: I think cards are an attempt to address this, but your point is well-taken. When talking about design, people [optimise] for the worst-case scenario.

At Foursquare, we had artificial throttles in place to simulate different layers of slow data connections – so it was certainly something we thought about. But most startups in America do not expect their primary use case to be populations in a second or third-world country, so it's essentially a non-issue.

My guess is that data will improve in those countries before Western startups start targeting them. As for startups in Africa and Central Asia, I'm sure it's top of their minds.

How useful are habits in UX design?
TW: Habits are neutral in UX; whether or not your product needs habits, what kind, and at what point, is entirely dependent on what you're building and what you want your users to do.

iOS designers stick to a strict UX pattern library so that the habits you form have the same result every time: there are no surprises, and you never make crucial mistakes. Car interface designers have similar sets of equally thoughtless, easy patterns: being able to turn on headlights and windshield wipers without taking one's concentration off the road is paramount.

But when the interface looks the same, but behaves differently, it's annoying. I've tried to tap or swipe my Classic Kindle screen a dozen times, even though it's not a touchscreen. It's not Amazon's fault – the Kindle has been around almost as long as the iPhone – but when one new interface is great, people will want that level of interaction in other devices, too.

The way to create habits in UX is to make sure the same thing happens every time you perform an action. That's easy. The hard part is getting people to get into the habit of using your app or site.

MOBILE

DESKTOP-LAST: WHAT ARE WE SO AFRAID OF?

Illustration by Ben Mounsey

It may be the buzz-phrase of the moment, but is anyone really embracing mobile-first design? **Dan Tello** shares his views on how to tackle a multi-device future

As developers, we take great pride in our work. Our projects are well built and well designed. They're responsive. They win awards and please our clients. They're awesomely beautiful on cinema displays … but they're completely boring on your phone.

Why is it that in nearly every award-winning desktop experience we've seen or built this year, there is a breakpoint at which everything goes from immersive, revolutionary and rad to static, stacked and boring? I'd like for us to step back and honestly evaluate the state of the multi-device web.

Open up *Awwwards.com* or *theFWA.com* on your desktop and browse the latest hot designs the web has to offer. You'll find site after site of beautifully done desktop experiences. Open the same URLs on your phone, and if you don't get a message like: 'Not formatted for your device' (the first one I tried), you'll get a fairly tall, single-column view of everything you saw at a desktop width, but with less … experience.

In an industry where the term 'mobile-first' is ubiquitous, why are smaller devices still treated as second class? Mobile-first seems to be generally regarded as a positive thing in our industry, but when it comes time to start a project, why are we, our team and our clients so afraid of actually putting it into practice?

I sent out a survey to get a feel for people's hesitations about mobile-first. An overwhelming amount of responses implied an underlying belief that the decision to build mobile-first is primarily related to the amount of mobile traffic the client is currently receiving. "Less than 50 per cent mobile traffic? Desktop first." Stop. This logic is not sound.

Low mobile traffic is not necessarily indicative of low mobile interest. Is the current amount of traffic related to the nature of the content, or is it simply because the existing experience is hardly usable on a small screen? It could be either, but it's important not to overlook this distinction. There may be a large potential mobile user base just waiting to be tapped into.

PITCHING FOR THE FUTURE

The data shows where we've been, not where we want to go. Low usage today isn't necessarily indicative of low usage in the future. As former ice hockey pro Wayne Gretzky once said (and as Steve Jobs often quoted), "Skate to where the puck is going to be, not where it has been."

Say a sales lead comes in and you pitch your strategy to the client, whatever it may be. Three months later, you've won the project, and in another nine months, you deliver your beautiful new baby into the world. From your initial sales pitch (12 months ago), to the end of the lifetime of what you built (around three years) is a total of four years. Where is the puck going to be in four years' time? Will you still be glad you decided to budget time into supporting IE8 instead of pushing the mobile experience?

The implied assumption that designing for small screens first will somehow be detrimental to the

desktop audience is simply false. By designing for small screens, your content and UI has the potential to become laser-focused, delivering your user exactly what they're looking for. There's no temptation to fill space with superfluous fluff when the space isn't there. 'Simple and focused' need not mean 'boring and limited'.

CREATING AN IMPACT

"Small screens are limiting." I often hear this argument from designers who are accustomed to wowing clients with cinema-sized Photoshop comps. And it's true – small screens are less impactful in a client meeting. But this isn't a limitation of technology, it's a limitation of client meetings. The truth is that there are far more opportunities for groundbreaking experiences on mobile devices than there are on desktop.

"It's not the technology, it's the imagination," says designer and mobile maven Josh Clark (@globalmoxie). We have touchscreens! With cameras! Gyroscopes, accelerometers, geolocation! All connected to the internet, to other people, and untethered to any location. We've barely scratched the surface of what we could do on mobile. This is exciting! Instead of presenting cinema-sized static comps, put an interactive prototype in the hands of your client and watch the excitement spread.

On mobile we have touchscreens! With cameras! gyroscopes, accelerometers, geolocation! Instead of presenting cinema-sized static comps, put an interactive prototype in your clients' hands, and watch the excitement spread

PROOF IT WORKS

There's nothing like using a well-crafted native app. The interactions are slick, everything is perfectly sized for your screen, and the content is focused and easy to get to. Why? Because it was designed specifically for the small touchscreen in your hand. Every interaction and piece of content has been carefully whittled down to its essence – in part, because there simply wasn't physical space to do otherwise. Your users love it.

Hold that thought. Now let's build a well-crafted desktop-only web app, assuming the same functionality as the native app described above. Because we built with the desktop experience in mind, our design assmues a wide, horizontal screen and things like a mouse cursor, and we have all the room in the world for navigation, content, headers, footers and asides. The site is beautiful and wins an award.

Now remove the '-only' suffix from the two (mobile/desktop) previous tasks. Assume that in both scenarios, we knew all along that we had to deliver experiences to mobile, desktop and everything in-between. Which starting point better positions you to move forward towards the best overall user experience? Do you wish more apps were like desktop websites, or that more desktop websites were like native apps? Which direction scales more easily?

With desktop-first, the initial design patterns that get wireframed, designed and approved assume a large screen, a wide aspect ratio, a mouse and a keyboard. This can result in patterns that simply don't translate well to a tiny touchscreen. Long select lists, large tables, mega-menus, hover states and content length suddenly become problematic.

We do our best to jam it all in there, but it often means burning endless hours on complex workarounds. Elegance is sacrificed for the sake of time and budget. Nuance and intention are lost, and solutions have to be hurriedly shoved in place if they are to make it in before launch.

On the other hand, to translate a mobile-first design to tablet and desktop sizes, all you might need to do is adjust the layout with some CSS to fill the space. There is no 'squeezing' of content, design or interaction into a place where it doesn't fit, and no inherent need to change the markup when moving from smaller to bigger. Additionally, you're starting with intensely focused content. You are in a better position to make informed decisions about whether or any additions are necessary. The translation is smooth, and you get that 'app' feeling for free, because that's what you started with.

Now we've put our mobile-first fears to rest, it's time to bring some much needed life to mobile browsers everywhere. Beginning small just might be the key to making something truly big. I can't wait to see what we build.

PROFILE

Dan (@dantello5) is a Washington, D.C. area senior frontend developer at Viget. He can be found blogging on *viget.com/extend* and pushing code at *github.com/greypants*

net SURVEY

WIN £250 of Amazon Vouchers

ARE YOU A WEB DESIGNER OR DEVELOPER?

Have your say! Whether you're a beginner or professional, take part in the Web Design and Development Industry Survey 2016

PLUS! FREE EBOOK WORTH £9.99

Complete the survey to claim your free handbook

WWW.CREATIVEBLOQ.COM/SURVEY

For full terms and conditions, please refer to: *www.futureplc.com/survey-prizedraw-terms-and-conditions*. All entrants must be aged 16 or over. Closing date 15 February 2016

BUSINESS

STYLE GUIDES FOR THE DIGITAL AGE

In an increasingly multichannel world, why are brand guidelines still focused on print? **Tom Dougherty** looks at how to write the rules for a digital era

I know what you're thinking – that headline sounds like something you should have been reading 10 years ago, right? But how many times have you started working with a new client and been sent the outdated brand guidelines PDF to review? We've all been there. 98 pages of print-focused layout rules, logo exclusion zones, CMYK references … Then right at the back, you'll find it tucked away: 'Digital usage'. Normally consisting of two pages of inadequate information, leaving you asking many questions.

ARE THEY REALLY NEEDED?

It seems crazy that in this day and age the digital environment is still considered secondary to above-the-line communication when it comes to providing guidance on how a brand should be represented. Digital channels are at the heart of most brand communication nowadays, and are where the highest volumes of people will encounter a brand. Providing guidance on how that brand should live and breathe online is essential.

Most creatively minded folk might argue that the lack of rules and guides is no bad thing, but from a brand's perspective it can be damaging to have so much inconsistency across its digital output. At Delete many of the global brands we work with have a roster of digital agencies they often work with. This only reinforces the need to establish some baseline guides for people to work from.

WHY IS DIGITAL NEGLECTED?

So why is digital commonly neglected? Brand guidelines have traditionally been the output of branding agencies and design firms where digital communication is not a focus. This means the digital 'section' of a style guide is often not fit for purpose for the multi-device, multichannel world.

Just think about the multitude of places that exist online where a brand can be visually presented: a web page, mobile page, mobile app, email footer, Facebook ad, display banner, social post … the list is endless. Of course, we're not just talking about a static presentation either. There's animation, audio, transitions – none of which can be delivered in a PDF document.

It's up to us, as digitally minded creatives, to challenge the traditional approach. There

At the back of the brand guidelines PDF, you'll find it: 'Digital usage'. Normally consisting of two pages of information, leaving you asking many questions

will always be an opportunity to have this discussion with your client early on. A few simple questions should soon make them realise work is required: "How should the logo behave at different responsive breakpoints? Is there an icon fallback for small screens? What are your web typography standards?" Clients will likely be looking to you for guidance on these issues.

Producing digitally focused guidelines is an opportunity for you to grow your relationship with a client, as you will be essentially operating as a digital brand guardian. If you're involved in a redesign project or even a one-off campaign, it's worth pitching this as a key deliverable.

So how should you approach these digital brand guides? Well, for a start they shouldn't sit inside a PDF document! It's essential the information is displayed in the medium in which it will exist. A simple HTML framework will allow you to categorise your elements clearly and provide code snippets where possible, to reduce the margin for error.

Things you should be considering include: logo usage (dimensions, colours, padding), responsive logo views, logo icons (app icon, favicon), RGB colour references, CSS typography styles, the standard masthead and footer, iconography, buttons (primary CTA, secondary), patterns (overlays, backgrounds), grid structures, form styles, animation examples, video opening and closing frames, social post examples and accessibility standards.

Providing examples of usage will also help any agency or external partner understand the dos and don'ts. Downloadable digital assets will also prove handy: PNGs, SVGs, font files and so on.

THE FUTURE

Ideally, rather than ending up with two sets of brand guidelines, you want to have the print-focused guides accessible from this web page, too. That way you ensure there is one central point for accessing this information, which can be updated as required.

Picture this – you start working with a new client and you receive the *link* to its online brand guidelines. A slick, responsive style guide containing all the basic rules, code snippets and digital assets you need. And then down at the bottom, you'll find a downloadable PDF containing a few pages of print rules.

Now that would be nice.

PROFILE

Tom is UX director and partner at Delete (*deleteagency.com*), where he works with clients including Red Bull, Expedia, IKEA and Unibet

PROCESS

RESPONSIVE REORGANISATION

Kevin M Hoffman explores how organisations are throwing out the traditional structures in favour of something more flexible

Responsive web design has not only changed how we design sites, it's also changed the way design teams work together. Companies are evolving to support what good responsive design efforts need: small cross-disciplinary teams, more iterative conversations, and smaller loops into production to test ideas.

I believe these changes mirror a larger trend in business: a move towards more organic ways of structuring people and processes. This is happening because responsive design is as much about designing for multiple contexts as it is about multiple screen sizes.

PYRAMID STRUCTURE

Organisations are historically structured in a pyramid shape. There's leadership at the top, then discipline managers (such as vice presidents of marketing, CTOs and so on), followed by the various teams they oversee. This approach has been around since the industrial revolution.

It allows manufacturers to scale and produce greater numbers of a product without sacrificing on quality. Chances are you already know this process; you probably call it 'waterfall'.

Responsive design requires us to design different things at the same time. We may be designing two or three experiences within the same code base, changing the hierarchy of content and functionality based on screen size and device capability. This requires a team to be flexible.

One way of doing this is to organically mix expertise within small teams, organised around goals as opposed to disciplines. When it works well, it offers the right expertise and accountability, and facilitates rapid collaboration. Smaller, mixed teams can more easily reach out for dependencies such as content, approval or domain expertise.

Because a responsive experience has the potential to reach audiences across different contexts, it changes how organisations collaborate. Responsible parties need to be more informed.

CASE STUDY

Here's an example from a company that should warm any geek's heart: Nintendo.

One way of making things more flexible is to organically mix expertise within small teams, organised around goals as opposed to disciplines

The site *play.nintendo.com* uses character stories, game tips and quizzes to create a seemingly never-ending path of content.

It also uses responsive design in an interesting way. On a smaller screen, it narrows the focus onto individual characters, and creates a sequence of stories and tips around those characters. On a larger screen, however, it emphasises content types over characters, and surfaces more choices per screen. This design choice isn't just based on the availability of more pixels. This is a deliberate attempt to adjust context.

Zeroing in on a long trail of character traits on smaller screens creates a more linear, addictive flow. This mobile flow caters for its particular context – restless kids at a restaurant table, for example. On desktop, they might learn 'how do I beat that level', but on smaller screens they fall into a loop of 'tell me everything there is to know about one thing'.

Changing a design to suit what is going on in someone's life is a key strategic benefit of responsive design. We can use our knowledge about device capabilities and a visitor's context of use to create appropriate benefits that go beyond just progressive enhancement.

TEAM EVOLUTION

The Nintendo website includes intellectual property, game content and key messages. The content has to be coordinated between product teams, marketing oversight, legal experts (because kids), and game designers. Not to mention all the staff tasked with working on the site itself.

To do this efficiently, smaller, more iterative conversations have to take place between these departments. This is a great example of the 'responsive reorganisation' I'm talking about – a situation where responsive design can be the impetus for better ways of building together.

This evolution of how organisations structure themselves is not unique to the web design industry, and has been well documented in business research (see this article by John P Kotter for starters: *hbr.org/2012/11/accelerate*). You may find something that will help your organisation approach responsive design collaboration in a whole new way.

Read the extended version: netm.ag/RWD-274

PROFILE

Kevin (*kevinmhoffman.com*) is the author of *Meeting Design*, and founder of Seven Heads Design

*ELEMENT QUERIES

NEW WAYS TO BE FLEXIBLE

Dan Donald considers the benefits of responsive designs in which each element refers to the width of its container, not the viewport

We've all been figuring out what responsive design really means to us as we work through all kinds of projects, coming across new challenges on each. Often, the current thinking on a topic will have moved on between one project and the next. One such area of rapid change is the basis on which the dimensions of each element on a page should be calculated.

BEYOND THE VIEWPORT

While working on a hack with Al Jones (@itasljones) a couple of years ago, we got to talking about the fact that all we seemed to do was refer to the width of the viewport. What we wanted to achieve was something very flexible and reusable, but conventional approaches to responsive design didn't lend themselves to working this way. Instead, we wanted to create modules that would refer to their parent/containing element, treating it much like a miniature viewport.

At the time, there were a few other people talking about similar things. In 2011, Andy Hume had produced some example code for what he termed Responsive Containers (*blog.andyhume.net/responsive-containers*). That set the cogs turning in our minds. Maybe there really was something worthwhile in this idea of an element or module referring to its parent or containing element?

ENTER ELEMENT QUERIES

We produced a rough-and-ready demo to explore how this process might work. What we found was that although it initially appeared to complicate things, it actually provided us with a lot of freedom. You have the choice of when you refer to the parent element and when you refer to the viewport. You could even nest what I called 'trigger points' (the equivalent of media queries in our demo for change relating to the parent element) within media queries, allowing for quite precise control.

This notion of relating to the parent element has become known as 'element queries' and since the time of Andy's demo and our hack, a number of other people have written about them and the ways in which they might be employed, and created new demos and polyfills.

In some ways, it feels like element queries are in the same place responsive images were

Photograph by Dan Davies

Let's hack around with the idea of element queries. Between us, we may learn enough from our experiments to make them a browser-native reality

a few years ago. We have an idea about what we want to achieve and ways to simulate this behaviour, predominantly through JavaScript, but no standardised syntax or native implementation. Can we get to a point at which this ability is available to us natively? It's hard to know, but Tab Atkins has a blog post entitled '2014 State of Element Queries' (*xanthir.com/b4VGo*) that summarises the current thinking.

THE DEBATE NEEDS YOU

One thing we can all do to help is play. Maybe you have a project on which element queries could work well. Spend time trying some of the existing code, or find your own solution. Let's find some clear use cases where element queries really work. In doing so, we may also help to resolve problems of implementation, such as the 'circular dependency issues' discussed in Tab's post.

To me, the ideal candidate for element queries is responsive images. An image should refer to the space available to it rather than the entire viewport in many use cases, shouldn't it? Of course, knowing what an image should do and actually making it do it are two different things. I'm sure that if this were easy to implement, it would be part of the solution already. On top of that, we have more than one problem to solve in regards to responsive images!

Reading through blogs posts on the topic, I get the feeling we may need to discuss how to divide the work between HTML and CSS. I'd prefer that no description of how a module I've created should behave appeared in my markup, as with media queries – but I get the sense that it may not be so clear cut in this case.

I encourage you to find time to hack around with the idea of element queries, creating a module that adapts not just to the viewport but to the dimensions of its containing element. Between us, we may be able to learn enough from our experiments to make element queries a browser-native reality.

As designers, this process will give us the opportunity to expand how we perceive flexibility on the web. And as developers, it will enable us to guide the evolution of the modules and elements we create. Together, we may be able to build a web that works better for us all.

PROFILE

Dan is frontend and interaction lead at McCann Manchester. Sometimes he runs @speaktheweb. When not doing web stuff, he makes 'rough alt-rock noise/music' in @markof1000evils

TESTING

HOW DO YOU MANAGE CROSS-PLATFORM TESTING?

To test responsive designs, do you set up your own test suite, visit an Open Device Lab or resort to emulation? We asked the experts

LARA HOGAN

Engineering manager, Etsy

laraswanson.com

At Etsy, we have a variety of ways to test new features across platforms, and it's up to the designer or developer to choose their own workflow. Most tend to use emulators as they're developing, then when the product gets close to production-ready they'll use our device lab (*netm.ag/etsydevicelab-254*). We've got a variety of operating systems, versions, and screen sizes to test with.

VILJAMI SALMINEN

Lead frontend designer, Adtile

viljamis.com

I believe in testing on real devices. Software simulators can be useful, but in the end they can only do that: simulate the experience. At home, I use a setup with six physical devices running continuously when I'm designing something. I also tend to visit the nearby Open Device Lab a few times to make sure that everything works on a wider range of devices.

JEN MYERS

Developer and instructor, Dev Bootcamp

jenmyers.net

For me, cross-platform testing always goes much more smoothly if I spend enough time in the beginning carefully considering how easily the design and content will adapt across different devices, so I always think of that as the first step of my process. Once I've developed a solid enough foundation, I use browser emulator services to see what my site looks like in other operating systems and browsers, and a responsive testing plug-in to test different screen sizes. I also tend to test on the actual devices that I have access to. However, I still think the most important factor is the initial design and content planning.

ADAM ROBERTSON

UX director, fffunction

fffunction.co

Here at fffunction we use a mixture of physical and software solutions for cross-platform testing. We make really good use of the virtual machines available at Modern.IE, which we then run through VirtualBox. We also have a physical device lab, which we try to keep as up-to-date as we can. All of our builds go through these devices to ensure that the mobile experience is as good as it should be. There is no substitute for testing your work on actual devices.

NEIL RENICKER

Developer, Sparkbox

neilrenicker.com

Since the device landscape is so vast, there isn't a one-size-fits-all answer. The truth is that you need a combination of tools to do the job. First, default to using physical devices if possible. The electronics store works great for this if you're on a budget! Second, use a virtual machine like *virtualbox.org* for testing old versions of IE. Third, use an emulation service like *browserstack.com* for all the other edge cases. That isn't foolproof, but it will get you 80 per cent of the way!

DAVE OLSEN

Web developer, West Virginia University

dmolsen.com

I address the majority of my cross-platform look-and-feel issues by using content-based breakpoints. Simply forgetting that device widths and orientations exist simplifies testing immensely. Functional testing requires real devices and a device lab – it's tough to emulate the size of your finger or the speed of a network. Not everyone can build one, so check *opendevicelab.com* to find a shared device lab near you.

TOM MASLEN

Web developer, BBC News

tmaslen.com

At BBC News we have a drawer full of many different devices. We prioritise testing based on the popularity of each device, so desktop Chrome, Safari on iOS 6 and iOS 7, Chrome on Android 4.x and the stock Android browser on Android 2.3 make up our primary test group. As a minimum, you should invest in the most popular devices – iOS 7, iOS 6, Android 4.x and Android 2.3 – then offset that cost with emulators and VMs.

FURTHER READING

THE BEST PLACE TO START

If you read just one article about testing responsive sites across mobile devices, we'd recommend Jeremy Keith's blog post 'Left to our own devices' (*adactio.com/journal/5433*). Don't be put off by its (relatively advanced) age: it offers a great foundation and links to many current resources.

TOP TESTING TOOLS

Building a responsive site can lead to frontend code becoming increasingly fragile. 'Top tools for testing your responsive sites' (*netm.ag/254-toptools*) is written by two BBC technical leads and explores the tools used by the Corporation to ensure that its code behaves consistently across devices.

SYNCHRONISED TESTING

Synchronised cross-device testing is an efficient way of automating tests across a slew of browsers and devices. It solves two perennial problems: keeping devices in sync with the URL and ensuring that interactions are consistent across tests. Check out Addy Osmani's article at *netm.ag/synch-254*.

PROCESS

WHAT'S YOUR APPROACH TO PROTOTYPING?

Work everything out with a pen and paper, jump to a specialist tool or crack out your HTML, CSS and JS skills? The pros share their personal techniques

JON HADDEN

Founder, NiceUX

niceux.com

Remain flexible and consider the variables each project brings. If my intention is to quickly communicate a concept or rapidly test IA, I sketch or build a clickable PDF. For more complex nuances like data binding or responsive content structures, I may end up writing HTML, CSS and JavaScript. Ultimately, I want to effectively answer any questions I had before prototyping, as efficiently as possible.

SCOTT HURFF

Product design specialist

scotthurff.com

When I create a prototype, I factor in the following variables: how far do I have to go to communicate what's in my head, who's the audience and how much time do I have to complete it? The goal of a prototype is to vet whatever's in your head in the shortest possible time frame. Sometimes ideas can be vetted with a pen and a paper; other times, a quick-and-dirty iOS app.

BRAD NUNNALLY

UX designer

onestraythought.com

My greatest prototyping tool is a pen and notebook. These allow me to explore and play – if I were to start by using a particular tool, my thinking would naturally be constrained to the widgets available. Once a design is fairly solid, that's when I move into a tool like Axure for refinement. Working on paper allows me to focus on getting the behaviour right initially, rather than the layout. When presenting the prototype to a client, I show the entire process, including those early sketches. This makes the client feel more comfortable about the product because I'm able to show that every aspect was thought through.

DONNA LICHAW

Independent consultant

greatnorthelectric.com

I have a background in documentary filmmaking and have spent years making non-narrative things easier to consume by turning them into narratives. This applies to prototypes, too. The easiest prototype you can create is a narrative arc, and the IA should be the thread running through it. Prototypes should have plot points, engage the audience with a hook or problem as soon as possible, and keep them engaged over time.

GAVIN WYE

UX designer

gavinwye.com

The tools I use change depending on the problem I'm trying to solve. Sometimes it's a sketch, sometimes it's a rough visual but most often it's in code using CSS frameworks and tools like Mixture, Jeykll or Hammer for Mac. This way we can get something in front of users and stakeholders – in a browser on the device they choose – quickly. If you're working on a single product for a long time, it's a good idea to build your own framework and use this code to prototype. Take code from the developers, 'hack' it to meet your needs, then pass it back for refinement. Working like this gets you closer to developers, and has the advantage of baking progressive enhancement into the design process, something we need to do much more of. Too often, designers only consider the perfect outcome for their design. The web is not perfect.

WILL HACKER

Senior UX professional

willhacker.net

Keep in mind that prototyping is part of a larger design process, not an end in itself. If you want to test whether a design is usable, a higher-fidelity interactive prototype created in Axure or HTML may be appropriate. But if you are testing terminology to flesh out the IA of a mobile app, paper prototypes or something simple using Prototyping on Paper could suffice. The most important thing is to understand what you hope to achieve before making any decisions about the fidelity or interactivity of your prototype.

YAEL LEVEY

UX designer, Bad Robot Design

badrobotdesign.com

Prototyping is a great way to communicate with the rest of your team, as well as the end client. For the heavier-duty projects, Axure is a powerful tool. If I am working closely with other disciplines, lighter-touch tools like Flinto and even Keynote are great for getting designs on screen fast. Using HTML, CSS and a bit of jQuery can also be good – I have found it immeasurably useful to learn the rudiments of frontend code.

RESOURCES

PROTOTYPE TO SUCCESS

In this brilliantly insightful talk (*netm.ag/reichelt-262*) the Government Digital Service's head of user research Leisa Reichelt speaks of how her approach to prototyping has changed, making for better organisation and implementation of ideas.

MAKE IT A PRIORITY

Before you dive into a project, testing your ideas for effectiveness is vital. Providing you with an insight into the functionality of your design, this tutorial from Daniel Bramhall (*netm.ag/priority-262*) explains why prototyping is something every designer should do first.

AN INSIGHT INTO APPLE

Have you ever wondered how the top dogs at Apple approach prototyping? Wonder no more, as this neat presentation (*netm.ag/apple-262*) showcases their initial methods so you can see how your technique matches up.

EXCHANGE

Practical advice from industry experts

FEATURING...

KAREN MCGRANE

Karen is the author of *Going Responsive*, a new book on RWD
w: *goingresponsive.com*
t: @karenmcgrane

BEN SEYMOUR

Ben is the author of the *Practical Responsive Images* pocket guide
w: *benseymour.com*
t: @bseymour

JASON CRANFORD TEAGUE

Jason is an author, trainer and the senior creative director at Capital One
w: *jasonspeaking.com*
t: @JasonSpeaking

PAUL ROBERT LLOYD

Paul specialises in interaction design and frontend development
w: *paulrobertlloyd.com*
t: @paulrobertlloyd

TOP QUESTION

Which is good to start with, mobile-first or desktop-first?

Naomi Takeuchi, Witney, UK

Type talk Jason Cranford Teague suggests designing around type elements, as seen on *burningofcolumbia.com*

JCT: What we understand by 'mobile-first' is really a overstatement of the original intent. When Luke Wroblewski coined the term, he was really thinking about how content that can be sacrificed for mobile devices is likely not critical for the desktop sites either. So, you should start by thinking about the content needed for the smaller screen, then evaluate anything you think is needed for larger screens.

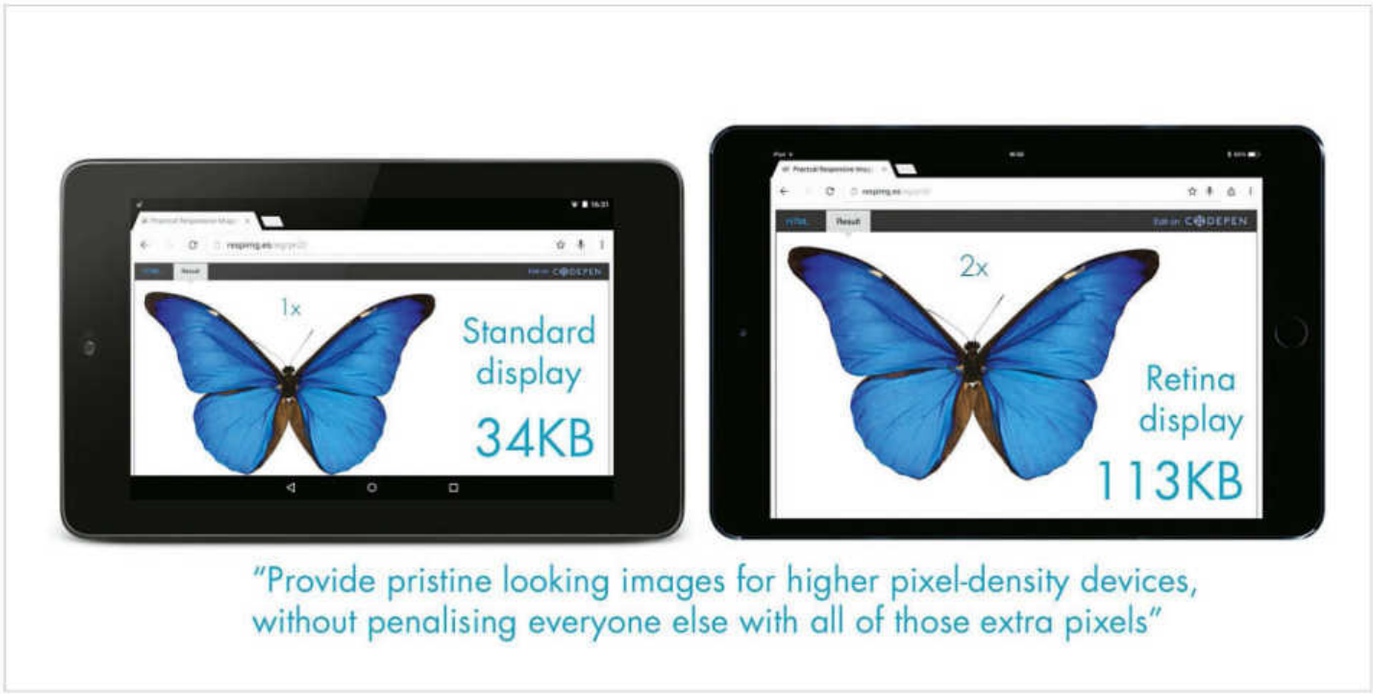
For my designs, though, I prefer to design typography-first. As so much of the page is text, I think of this as the primary design element and elaborate the design from there, adjusting the typography as needed to respond to the screen context.

LIQUID AND ADAPTIVE

What about this whole new-fangled 'liquidapsive' thing?

Michael R Bernstein, New Mexico, US

KM: Liquid and adaptive are terms used to describe grids that are not completely fluid, as in a responsive design (visit *liquidapsive.com* for some demos). While responsive design is handled client-side, adaptive is also used to refer to server-side techniques. Adaptive content means sending different content to different devices, while adaptive designs serve different HTML to solve particular layout or design problems. 'Adaptive' can be



Tricky images Start off supporting resolution switching by adding the scrset attribute

used in so many different ways, it's a good idea to clarify that it's not a synonym for 'magic'.

TRICKY IMAGES

Responsive images look interesting, but also quite complex. Where would you recommend getting started?

Levi Kidston, Ohio, US

BS: You could start supporting resolution switching by just adding the `srcset` attribute with `x` descriptors (`1x`, `2x`) to your existing `<img>` tags. This already has solid browser support and enables you to provide pristine looking images for higher pixel density devices, without penalising everyone else with all those extra pixels.

Liquidapsive.com Visit this site for handy demos

Using `srcset` with the width descriptor enables you to form an image candidate list that provides the browser with a list of different-sized versions of each image, so the User Agent can choose the most appropriate one.

In both cases you must still include an image `src`, so non-supporting browsers will continue to work the old way. For broader browser support you can use a polyfill such as Picturefill (*netm.ag/piturefill-271*).

HUNDREDS OF THOUSANDS

Some of my clients have hundreds of thousands of images on their sites. Do you have any tips for dealing with the image size/resolution variants at such volumes?

Zach Yuranigh, Broughton, AU

HJ: You can automate the production of the image variants using desktop application workflows or via command line scripts, but at larger volumes these approaches can introduce delays. A bigger concern is that by 'forking' your content, you also risk ongoing maintenance, management and synchronisation headaches. Server-side Dynamic Imaging solutions are becoming more mainstream. These typically take a single master asset and generate almost unlimited variations of size, format and quality at request time. You are then free to experiment, knowing that the Dynamic Imaging solution can provide whatever variants you require, on demand.

You could also consider building your own, or using an SaaS option. Some services offer additional benefits such as 'point of interest' for guided art direction. I would recommend coupling it with a suitable Content Delivery Network for scalability and reliability.

KEEPING UP SPEED

What's the best way to do responsive without sacrificing page speed?

Jan Kenneth Regala, Makati, PH

JCT: The best way to keep pages fast is through solid coding, using semantic HTML, and relying on CSS for the design rather than using images. Anything that's not actually graphic content (photos, illustrations and so on) should be added in the CSS. You should also move any non-immediate CSS code so it's loaded in the footer of the document. Code that is not immediately needed to render the page on load (e.g. user interactions like hover, focus and

Speedy pics Responsive Issues Community Group

active) can be loaded last. This doesn't affect the page appearance, but allows important content to load faster. Another approach is to lazy load images (see following question).

LAZY LOADING

What's your technique for lazy loading responsive images?

Marcel Weber, Nusplingen, DE

JCT: Lazy loading images is a 'Just in time' method of adding image files to the page that are not loaded or rendered until they are actually needed. This can greatly speed the apparent load time of the page, since those big image files don't have to come down the pipeline all at once. There are several methods for this. Most rely on backend tech or JavaScript, and none of them is perfect – they often require that in the future, we'll all be using the `<picture>` tag with separate sources based on media queries, coupled with JavaScript to load them as needed. For more details, see the Responsive Issues Community Group (*ricg.io*).

UPDATING SITES

What's the best way to design a responsive site when there's an existing non-responsive site?

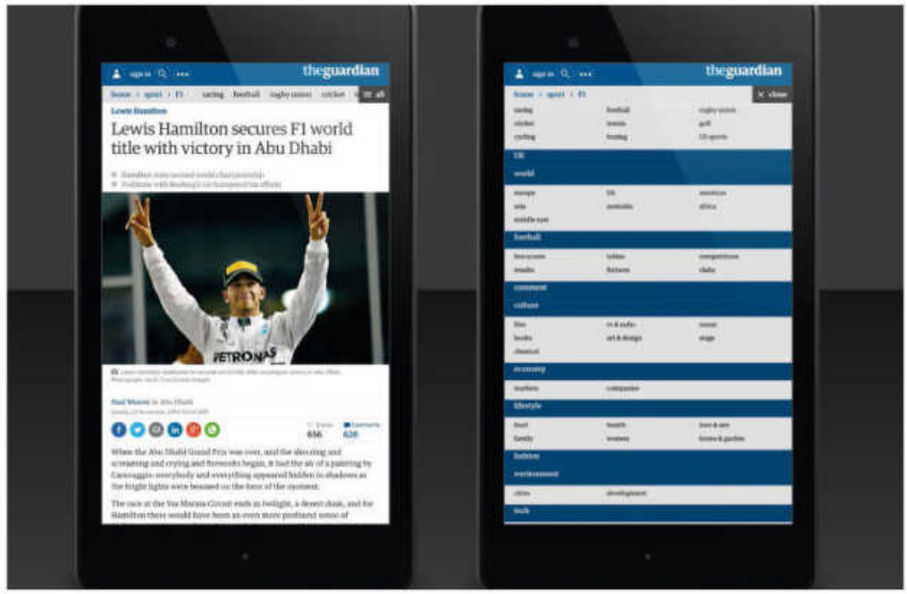
Jordan Carroll, Belfast, UK

JCT: The short answer is: start from scratch. The longer answer is that it depends on how the HTML of the non-responsive site was coded. If we're talking lots of `divs` or – Cthulhu forbid – tables, then refer to my short answer. If the original site uses well-coded, semantic HTML, adding responsive CSS shouldn't be too difficult. It may still require some tweaking of the HTML,

TOP QUESTION

What is your recommended method of handling drop-down menus on tablets and smartphones?

Jordan Simpson, Birmingham, UK



Condensed *The Guardian*'s mobile site prioritises page signposts, but all sections can be accessed with one tap

PRL: Before thinking about a menu's presentation or behaviour, first ensure that the items it contains are relevant to users, whatever the breakpoint. Thinking mobile-first rather than trying to adapt desktop design patterns downwards can help prioritise and rationalise menu items, too. The *Guardian*'s navigation is a good example. Its intriguing design prioritises page breadcrumbs at smaller screen widths, with horizontal scrolling allowing you to access sibling and top-level section items. An 'all' link provides a safety net, giving you access to every section with one tap, starting with the section you are in (*netm.ag/guardian-264*).

A great resource for responsive patterns (including those dealing with navigation) is Brad Frost's 'This is Responsive' (*netm.ag/frost-264*), but new possibilities are being uncovered every day.

but the trick will be to find ways to keep important content at the top of the longer scrolling screen. The new CSS flexbox method (*netm.ag/flexbox-267*), which allows complete reflowing of content boxes and is supported by almost all modern browsers (including mobile browsers) is a great solution.

DASHBOARD TABLES

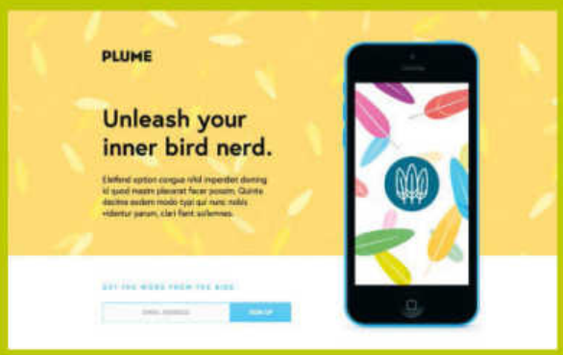
I deal a lot with dashboards, but can't find a good way of handling tables. Please help!

Joe Watts, London, UK

PRL: As no two tables are the same, there's no definitive solution. Simpler tables could be presented as a list at smaller breakpoints, while more complex tables may require a change of orientation so that column headers appear fixed on the left, with data in a scrollable area to the right. Or maybe you allow users to choose which columns to hide/show.

Jason Grigsby has a good summary of the options available – and what you should consider before choosing one (*netm.ag/table-264*).

PROJECTS



View source files here!

All the files you need for this tutorial can be found at: *netm.ag/susygit-274*

ABOUT THE AUTHOR

RAY VILLALOBOS

w: *raybo.org*

t: @planetoftheweb

job: Staff author, Lynda.com

areas of expertise: Full-stack design and development, frontend design, JS, AngularJS, Sass, Bootstrap

q: what gadget would you have trouble parting with?
a: I still have my very first 512K Fat Mac. It still works and I'm never letting go

SASS

FLEXIBLE LAYOUTS WITH SUSY AND BREAKPOINT

Ray Villalobos explains how to create semantic, flexible and backwards-compatible layouts with Susy and Breakpoint

Creating responsive layouts can be challenging because of the maths involved, so it's common for designers to turn to frameworks and/or Sass to simplify the process. A lot of frameworks are based on a 12-column grid, but a responsive site doesn't always fit into this structure. Here, I'll show you how to use two Sass extensions to create truly flexible layout that transcends the 12-column grid.

THE PROBLEM

Designing responsive layouts can be tough, as it involves calculating the width of containers, rows, columns and gutters at different breakpoints. Frameworks can help by creating presets for common breakpoints. Bootstrap 3, for example, gives us a 12-column grid with four media query breakpoints. You then use classes that cause your content to take up a certain number of spots in the grid, and the gutters always take up 30px.

This works brilliantly most of the time, but there are two problems. First, adding these classes to your markup can get a little verbose. Say you want a layout that uses all the columns on mobile devices, six of the 12 columns on small devices, and four on medium devices. That markup might look something like this:

```
<h2>Services</h2>
<div class="row">
 <article class="service col-md-4 col-sm-6">
```

VIDEO

Ray Villalobos has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/susyvid-274*

Bootstrap grid In a simple Bootstrap 3 layout with just three breakpoints (two shown above), there are classes that don't add semantic value

```
<img class="icon" src="images/icon-exoticpets.svg"
alt="Icon">
<h3>Exotic Pets</h3>
<p>We offer specialized care for reptiles, rodents, birds, and
other exotic pets.</p>
```

The second problem is that these classes add layout information to your HTML and make your code difficult to update, especially on a large installation. As your layouts get more complex, you might end up with some code that looks like this:

```
<div class="photo col-xs-6 col-xs-offset-3 col-sm-3 col-sm-
offset-1 col-md-2 col-md-offset-2 col-lg-4 col-lg-offset-0">
```

The larger issue is that you have little flexibility. Your framework should take care of the maths,

Susy's simple promise is to let you worry about the design while it takes care of the maths

because that's the hard part, but it shouldn't be dictating the metrics of your layout. Who made these frameworks the boss of you?

SUSY

Susy's simple promise is to let you worry about the design while it takes care of the maths. At its core it's a set of Sass mixins for calculating widths in a completely flexible grid system (*susy.oddbird.net*).

Start off by importing the library into your project via an import command: `@import "susy";`. This gives you access to Susy's grid framework, which couldn't be simpler to implement. In its most basic form, there are just two mixins you'll need to learn. First is the `container` mixin.

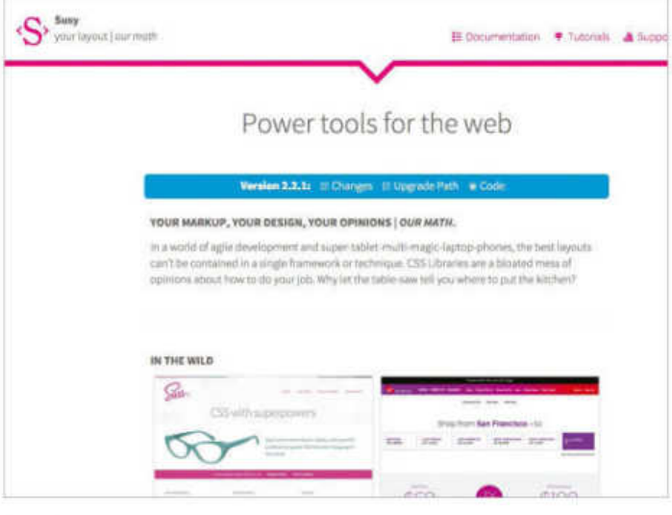
FOCUS ON

WHAT ABOUT FLEXBOX?

Flexbox is the future of layouts, so why not use it instead? Most of us should use progressive enhancement to support older browsers. If you need to support IE8 or 9, it's better to produce code that lays things out with a traditional model. The reason most people use flexbox is because it can be simpler to understand than the box model, but a layout system like Susy can provide a flexible and responsive grid that adjusts to different breakpoints *and* is relatively straightforward.

What's more, some users have reported slowdowns in performance at scale when using flexbox for laying out pages. It has also seen a lot of changes since its inception, so users are required to to work with several implementations of the spec, plus have a strategy to insert browser prefixes for older browsers (*caniuse.com/#search=flexbox*). In a serious progressive enhancement implementation, you're still going to need to have a strategy for dealing with legacy browsers.

Although tools like PostCSS and other are starting to simplify this process, using Sass with Breakpoint and Susy is much simpler to learn and implement. They also offer the backwards compatibility you need, without the hassle of dealing with flexbox plus the box model. Learn about flexbox and experiment with it for the time being, but it might not be the right solution for production layouts at the moment.



Enter Susy Susy is a series of mixins for Sass that takes care of the maths for any layout grid, so you can focus on the design elements

IN-DEPTH

CREATE AN IMAGE GALLERY QUICKLY

Need to create an image gallery or quick grid? Susy comes with a gallery mixin that makes that a cinch – just target your `img` tags and use the `@include` gallery mixin. You can easily create a four-by-four grid of images using the following:

```
.gallery {
  @include clearfix;
  img {
    border-radius: 20%;
    display: none;

    @include breakpoint($small) {
      @include gallery(1 of 4);
      display: inline;
      &.hidesm { display: none; }
    } //breakpoint

    @include breakpoint($medium) {
      @include gallery(1 of 5);
      display: inline;
      &.hidesm { display: inline; }
    } //breakpoint
  } //img
} //gallery
```

With this layout each image takes up one of the four columns on the grid by default. I've also set up three different scenarios for my grid. On a mobile device, we hide the images to provide quick access to the content. At a `$small` breakpoint, we set up a four-column grid and through an additional class of `.hidesm` on the fifth image, we prevent the last image from displaying. At a larger breakpoint, we switch to a five-column layout and display the additional image in the fifth column.

This is by far the quickest way to create responsive image galleries with a minimal amount of code.

Percentage width The welcome section on this site uses Susy's flexible containers set to a percentage of the width of the viewport

Containers

Containers help you control how the width of an element adjusts to different breakpoints. With Susy, we can redefine the containers inside any element at any time, without having to add any additional classes to our HMTL code.

Let's say I'm laying out a different website. If I want to create a container within an HTML element, I can add an `include` within my declaration, like this:

```
#welcome {
 article {
   @include container(70%);
 } //article
}
```

This will make the item with a class of `section` work like a bootstrap container, without having to add a `.container` class into our HTML – meaning our HTML can be a lot more semantic.

```
<div class="scene" id="welcome">
 <article class="content">
   <h1>Welcome to the Landon Hotel</h1>
   <p>The original Landon perseveres after 50 years in the
heart of West London. The West End neighborhood has
something for everyone&mdash;from theater to dining to
historic sights. And the not-to-miss Rooftop Cafe is a great
place for travelers and locals to engage over drinks, food, and
good conversation.</p>
 </article>
</div>
```

In addition to making my HTML a lot more readable, this makes things easier to update. When we use classes like `content` and `scene`, it's easy to redefine what those elements mean in terms of layout, instead of having to think about how many classes we'll need to add in order to make our content behave the way we want it to.

Spans

In Susy you create rows and columns using spans. To create an element that takes up one of three columns, you can write something like this.

```
#usefulinfo {
 section {
   @include span(1 of 3);
 }
}
```

Any element in my layout can take up only the amount of space I need at that time

What's really great about this is that we never have to conform to a specific number of columns and then adjust the elements accordingly. Any element in my layout can take up only the amount of space I need at that time.

This really changes the way you think about columns. If I was using Bootstrap, I would write the above code as `.col-sm-4` since one third of 12 columns is four. With Susy, I don't have to think about how many units I want to span; I can simply specify the amount of space I need. When you're no longer thinking about the conversion to a specific number of columns, you can focus on what the layout should look like instead.

SETTING UP DEFAULTS

Of course, in any layout system it's good to have defaults, so we don't have to specify the size of our gutters in every instance. We do that by modifying a variable called `$susy` at the top of our Sass:

```
$susy: (
 columns: 12,
 container: 60em,
 gutters: 1/4,
 gutter-position: inside
);
```

Susy has a ton of defaults you can use to set up your default grid, but this basic set-up will take care of putting together a standard Bootstrap-like default. Don't forget, everything in Susy is customisable so you're never married to any of these, and can change them on a tag-by-tag basis.

The default grid will now have 12 columns when we use the `@include` command in the container mixin, and that container will lock on at 60em width, with gutters that are a quarter of the size of the columns. If we wanted to fit our earlier sections to this grid, we could write the declaration like this:

```
#usefulinfo {
 section {
   @include span(4);
 }
}
```

This means each section takes up four of the 12 columns. However, I think it makes more sense to be able to say that an element takes up 'one of three' columns. If you need to offset columns to a certain position, you can use this notation:

```
@include span(8 at 4 of 12);
```

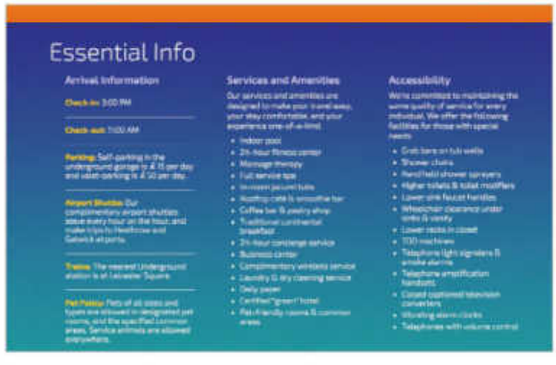
This lets an element take up eight columns, starting at the fourth position in a 12-column grid. So when you're creating a layout, you can focus on what your content needs to do instead of how the design fits into your existing grid.

PADDING COLUMNS

Another way to control the placement of your elements is by padding your columns. This adds a number of column spaces on either side of your columns. For example, you could move a column seven units to the right and pad it one unit from the left, like this:

```
@include pad(7,1);
```

This means that in addition to laying things out by thinking about positive spaces (how many columns an element should take up), you can do the reverse and create designs based on the spacing on each side of your content.



Span statements Using a simple @include span statement, we can set up each of the containers to fit into a custom, three-column grid

FOCUS ON

VISUALISING YOUR GRIDS

In Susy, you can add a special variable called `debug` to your settings. This will enable you to visualise the grid in your layouts. Let's take a look at how it works.

```
$susy: (
 columns: 12,
 debug: (
  image: show,
  color: rgba(196, 34, 111, .25),
  output: background,
  toggle: top right,
 ),
);
```

You can pass along four different parameters to these settings. First is settings, which will determine the type of visualisation you get. You can use `show` or `hide` to toggle the grid on or off, `show-colums` to show only the horizontal grid columns or `show-baseline` to show only the baseline grid. By default, the visualisation will display with a semi-transparent blue background, but you can modify the colour variable to change this.

Output controls whether the grid displays in the background of the element or as an overlay on top of the content. Finally, Susy can generate a toggle to display the overlay while you hover over an element. You use the toggle option to describe where you want to display the toggle.

The visualisation will show you the default grid, but you can modify the display of the grid on an item-by-item basis by adding a different include within your content, called `show-grid(4)`:

```
.container {
 @include show-grid(4);
}
```

It's great for previewing the grid while in development, but don't forget to turn this off as you publish to production.

MANAGING MEDIA QUERIES

To make things truly responsive, you need to be able to combine column set-up and design with responsive breakpoints. To do this, we can use another set of mixins from a library called Breakpoint (*breakpoint-sass.com*).

Breakpoint makes it easier to handle media queries in your layouts. It does so by simplifying the language you need to use to create media query breakpoints. Traditionally, you create these using a rule like this:

```
@media (min-width: 34em) and (max-width: 62em) {
 .container {
  ...
 }
}
```

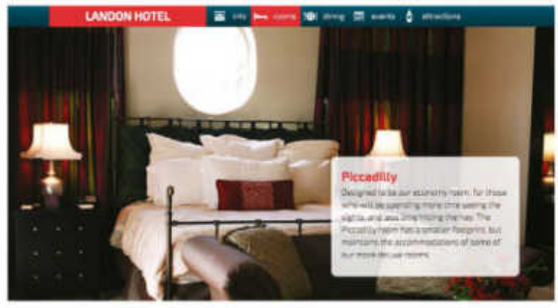
This creates a range of widths in which the declarations have an impact on your layout. Breakpoint encapsulates the calls into a common-sense mixin that's much easier to write:

```
.container {
 @include breakpoint(34em 62em) {
                ...
 }
}
```

The call is different too, because we can easily assign this inside an existing class. The great thing about Breakpoint is that it makes assumptions based on common layout needs.

Breakpoint rules are easy to learn too. There are just three things you need to know:

1. If you only include a single number in the breakpoint call, Breakpoint will assume a min-width media query call
2. If you include two numbers, breakpoint will assume you want to specify a range between the two numbers (as in the previous example)



Negative space Using the pad include, it is possible to lay out your content based on negative space

3 If you include two values and one is a string, it assumes you are sending the mixin a feature value pair, so if you want you can still send in orientation or any other special media query rule

Breakpoint has been folded into Susy since version 2.2.1. The Susy version works just like the Breakpoint mixins, but instead of calling `breakpoint`, you use `susy-breakpoint`. The same call would be made like this:

```
.container {
 @include susy-breakpoint(34em 62em) {
  max-width: 50%;
  margin-left: auto;
  margin-right: auto;
 }
}
```

BREAKPOINT AND SUSY

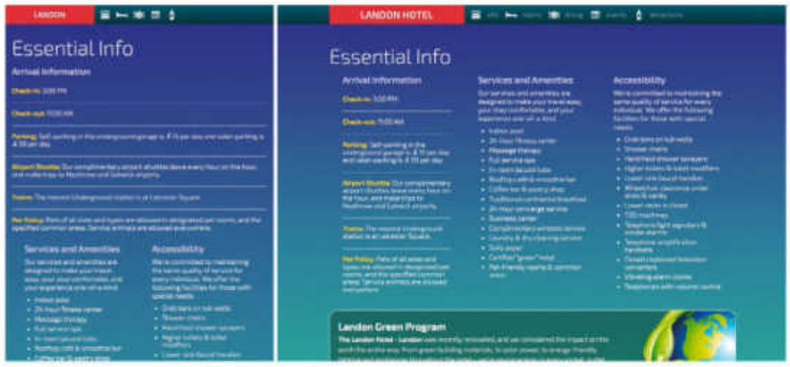
When you combine Breakpoint with Susy you get a flexible grid that can easily adjust to different media query declarations. Let's take a look at the HTML for the information section on a site.

```
<div class="scene" id="hotelinfo">
 <article id="usefulinfo">
  <section id="arrivalinfo">
  </section>
  <section class="checklist" id="services">
  </section>
  <section class="checklist" id="accessibility">
  </section>
 </article>
 <article id="greenprogram"></article>
</div><!-- hotelinfo -->
```

Combining Susy and Breakpoint, we can create media queries that contain different grid layouts. In my layout I have three different sections, but I want them to have different rules depending on the width of the viewport.

At these two different breakpoints, our layout behaves dramatically differently. In the larger breakpoint to the right, each section takes up three columns. In the smaller breakpoint, the first column takes up 100 per cent of the viewport, but the other two take up half of the viewport. This is what makes these mixins so powerful. The code for expressing these two layouts is concise:

```
section {
 @include susy-breakpoint(650px) {
  @include span(1 of 3);
 } //breakpoint
```



Make a move At two different breakpoints our layout behaves differently, but this is fairly easy to express in Sass with Susy and Breakpoint

```
 @include susy-breakpoint(450px 650px) {
  &.checklist {
   @include span(1 of 2)
  } //checklist
 } //breakpoint
} //section
```

First, we set up defaults for any element in a section tag. If those elements reach a viewport that's greater than 650px, they will occupy a three-unit grid, with each taking up a single unit.

If the layout is between 450px and 650px, the elements with a class of `.checklist` (the service and accessibility sections) will fit into a completely new grid with only two columns, and each element will take up one of those units.

The arrival information takes up 100 per cent of the grid between these two breakpoints. Notice we didn't specify what happens below 450 pixels. This is taken care of by the defaults – each section will take up 100 per cent of the viewport.

MASTERING LAYOUTS WITH MIXINS

What these two mixin frameworks give you goes beyond the code you use to create the layouts. The ability to express your layouts without having to worry about this grid or that grid changes the way your brain works when designing projects.

Don't get me wrong, I love frameworks. Bootstrap can help you create layouts with unprecedented speed. It offers a battle-tested grid that goes beyond layout to provide all kinds of CSS and JavaScript components to quickly handle common elements like tables, modals and forms.

What the system of design I've outlined does is change the language you use to describe a layout. It's a natural, backwards-compatible and easy to learn system that will change the way you think about designing websites.

RESOURCE

Eric Suzanne, the developer behind Susy, has a collected together a great list of tutorials and videos to get you started. Check it out at *susy.oddbird.net/demos*

ABOUT THE AUTHOR

WES BOS

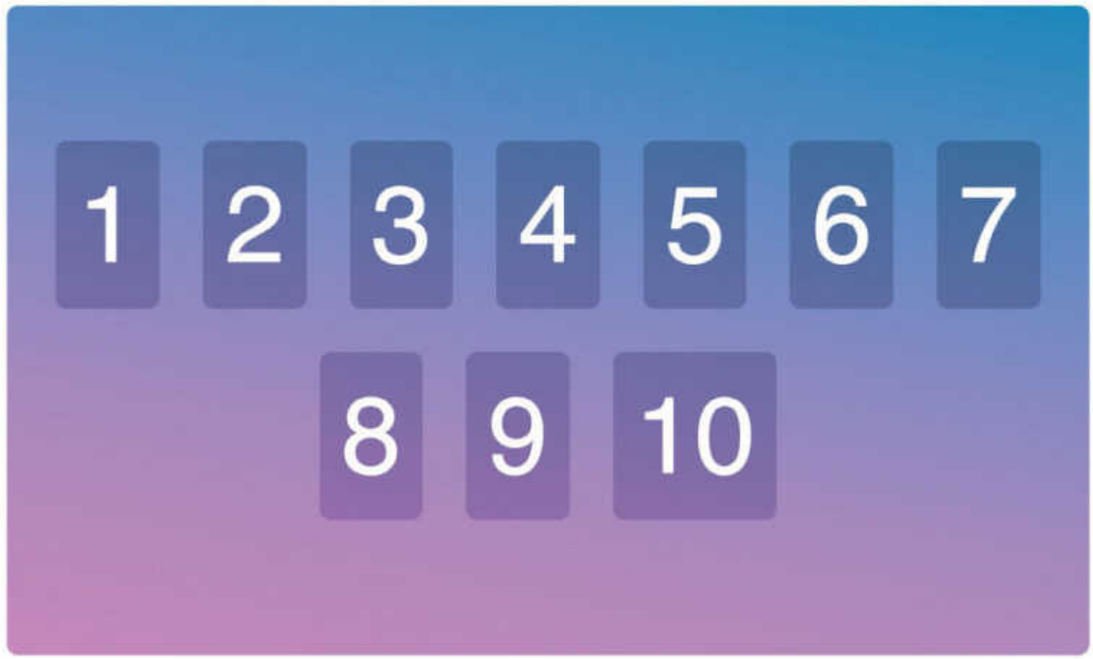
w: wesbos.com

t: @wesbos

job: Full-stack developer, author, speaker and instructor

areas of expertise: HTML, CSS, JavaScript, workflow, tooling

q: what was the first album you ever bought?
a: Hanson! 'MMMBop'!



FLEXBOX

SOLVE YOUR LAYOUT PROBLEMS WITH FLEXBOX

Wes Bos walks through how to use the Flexible Box Layout module to easily centre, align, scale and reorder elements on your web page

BROWSER SUPPORT FLEXBOX

Desktop	Mobile/tablet
45	47
9	8.4
34	8
12	4.4
42	

Have you started using the Flexible Box Layout module (*netm.ag/flexiblebox-275*) in your projects yet? Although the module has been around for some time, there have been two major barriers stopping frontend developers from using it in their projects.

First, until recently the spec was in flux and there wasn't great support for it. Today, all modern browsers support flexbox (see browser support widget for details). The second barrier is that it is pretty tough to grasp the concept of flexbox. While it is super-powerful, there are a lot of moving parts and it can be difficult to learn.

In this article, I will get you up and running with the fundamental concepts behind flexbox. Understanding these core concepts will open up a whole world of extremely flexible, easy to create layouts.

NEXT-GENERATION DESIGN

Flexbox is a next-generation tool to help you create layouts with CSS; whether you want to lay out a section of your website or display a grid of media elements. It enables you to easily align, centre, justify, scale and reorder elements on your page, without having to resort to nasty CSS hacks or fragile JavaScript dependencies.

Flexbox can replace floats, positioning tricks, inline-block layouts and even – shudder – table display layouts. If you have ever pulled your hair

out wondering why some seemingly simple layouts were difficult or even impossible in CSS, you are going to love flexbox.

FLEX ELEMENTS

The magic of flexbox is in the relationship between the parent 'flex container' and the children 'flex items'. In order to take full control of flexbox, you must put aside any previous ideas of floats, positioning and clearing. This is a totally new way of laying out your page.

Setting `display:flex;` on a parent element turns it into a flex container, and all of its immediate children will be turned into flex items. Once you have your markup set up, you can use one of the many available flex properties to create a layout.

Flexbox removes the need for nasty CSS hacks or fragile JS dependencies

Note: any of the HTML elements can be a flex container or flex item. Any `:before` and `:after` pseudo-elements you have on your flex container will be treated as children, and therefore first-class flex items.

ROWS AND COLUMNS

There are two axes in flexbox that control how the flex items on the page are laid out: the main axis and the cross axis. By default, flexbox is set up so the main axis goes from left-to-right (or the opposite, for languages that read right-to-left) and the cross axis flows top-to-bottom (fig 1). Before you go memorising that, note that this can – and will – all change with the `flex-direction` property.

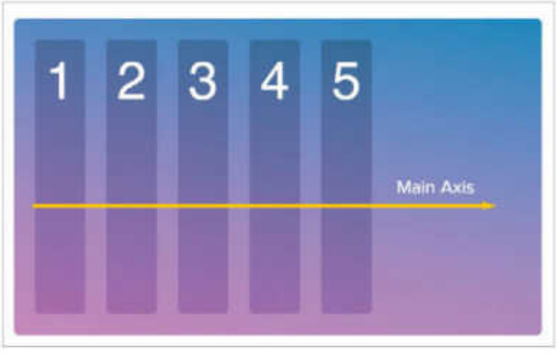


Fig 1 The default flex-direction set to row

FOCUS ON

FLEXBOX IN THE WILD

Bootstrap 4

Just a few months ago, the popular HTML/CSS/JavaScript framework Bootstrap announced its fourth version. Along with a whole load of new functionality, one of the things that had the community talking was the opt-in flexbox support.

Bootstrap comes with a grid, which helps designers and developers quickly create layouts by either appending classes like `.col-md-8` and `.col-md-4`, or extending a selector with `@extend .col-md-8`. In the first three versions this was done with floats, and Bootstrap handled everything to do with widths, margins and clearfixing parents. As flexbox is supported by IE10 and up, Bootstrap 4 will still use floats by default, but users will now have the option of turning on the flexbox-based grid instead, as part of the compile.

While this feature is still under heavy development, currently there is a `_variables.scss` partial that contains all of the settings related to your build. One of these is `$enable-flex`. Set it to `true` and recompile your Sass build to start using flexbox. You will continue to use your grid the same way you did with floats, but now you have all the added benefits flexbox has to offer.

React Native

Have you heard of React Native? It's a framework from Facebook that enables you to write iPhone and Android apps in JavaScript and React, and then have your app compile to native iOS and Android code. One of the most exciting parts is that the creators have chosen to use flexbox as the language that handles all the layouts in a React Native app.

React Native differs from other JavaScript-to-app frameworks in that it doesn't run any HTML/CSS on your phone. It simply uses the flexbox spec, which many developers are currently learning, to create the layout for your application. This is then converted into the equivalent styles on iOS and Android.

You can draw your own opinions, but I think it's never been a better time to know JavaScript and CSS!

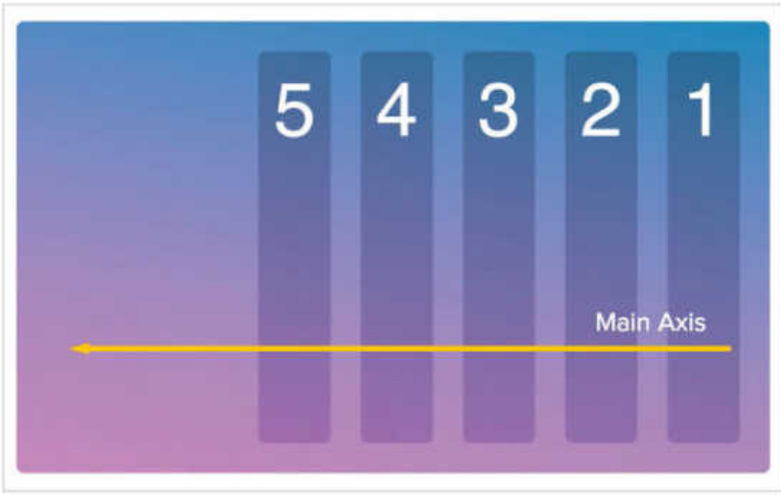


Fig 2 The main axis flipped with flex-direction:row-reverse;

By default, flexbox is set up with `flex-direction: row;` which means the items flow on the main axis from left-to-right, in a row. We can switch the main axis so it flows from right-to-left by using `flex-direction:row-reverse` (fig 2).

To switch both the main axis and the cross axis, we change the layout to `flex-direction:column;`. This will alter the main axis and flex item flow from left-to-right to top-to-bottom, in a column (fig 3). We can also start from the bottom and move up by flipping the main axis with `flex-direction: column-reverse`.

CENTRING ITEMS

One of the best things about flexbox is that it allows you to align your content in any way you please – even vertical centring is an absolute breeze! There is often confusion surrounding flexbox alignment, because there are three different properties that we use to modify the alignment of our flex items.

One reason for this is that these properties align the items along the main and the cross axis. So instead of asking, 'how do I centre something vertically or horizontally?' you must first establish which direction your axes are pointed, and then figure out which CSS property to use to properly align and centre your flex items on them.

In the next few examples, I'll be attempting to perfectly centre my items. However, you should know that each of these properties has a number of alignment options. For a full list, I recommend keeping this CSS-Tricks flexbox reference handy: *netm.ag/guide-275*.

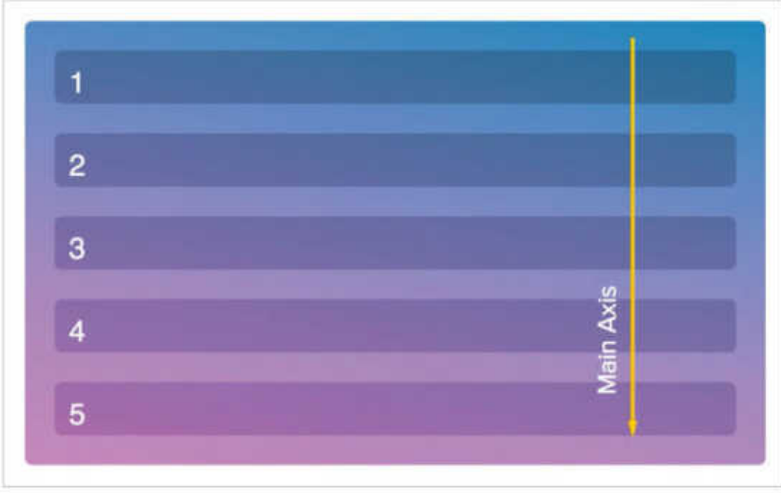


Fig 3 The main axis switched from horizontal to vertical with flex-direction:column;

THE CROSS AXIS

By default, flex items stretch across the entire flex container along the cross axis. If we want to centre items along the cross axis, we can use

There are three properties we use to modify the alignment of our flex items

the `align-items` property on our flex container and set it to `center` (fig 4).

In addition to `stretch` and `center`, we can also use `flex-start` and `flex-end` to anchor the items at the top and bottom respectively. Finally, we have `baseline`, which will align the items along the bottom (or baseline) of your text. This is extremely helpful when you are trying to align items with varying font sizes.

JUSTIFYING ALONG THE MAIN AXIS

Now we have the centring working along the cross axis – top to bottom in our case – we need to get centring working across the main axis. For this, we use `justify-content:center` (fig 5).

Just like with `align-items`, we can also use `flex-start` or `flex-end`, as well as `space-between` and `space-around`, which will evenly divvy up the remaining space between the elements. This is super-useful when working with grid layouts that don't add up to 100 per cent of the margin and widths.

Just remember, if we switch from the default `flex-direction:row;` to `flex-direction:column;`, the main

axis can change from left-to-right to top-to-bottom. When we switch to column, `align-items` becomes the horizontal alignment while `justify-content` becomes the vertical alignment.

ALIGNING MULTIPLE LINES

While `align-items` and `justify-content` work great when you have a single row or column of content, things get a little trickier when you're dealing with multiple lines of content as a result of using `flex-wrap:wrap;` on the flex container.

`align-content` works just like `justify-content`, but kicks in when we have multiple lines of content. By applying `align-content:center;` we can ensure that the lines will anchor in the middle of the cross axis and centre their elements from there.

Just like with `justify-content`, we can also use `flex-start`, `flex-end`, `space-between` and `space-around`. However, this time they refer to the space in-between the rows or columns of content, and not the flex items themselves.

Now, with just four lines of CSS, we have a bulletproof way of vertically and horizontally centring all direct children of a flex container.

```
.container {
        align-items: center;
        justify-content:center;
        align-content:center;
        flex-wrap:wrap;
}
```

FILLING THE SPACE

So far, everything we have learned about alignment has to do with the flex container and how it aligns its children. With `align-self` it is possible to override the `align-items` property set on the flex container by individually setting `align-self` to `flex-start`, `flex-end`, `center`, `baseline` or `stretch`.

Another often misunderstood part of flexbox is how to work with grow, shrink and basis values. It's helpful to once again throw away any ideas of pixel-perfect grids and embrace that flexbox is, well, flexible.

Each flex item can be assigned a `flex-grow`, `flex-shrink` and a `flex-basis` value. With these values we can indicate our ideal sizes, and then specify how the items should act in situations where there is extra, or not enough, space. From there, the items will just figure it out for themselves.

I like to think of these properties as:

- `flex-grow`: How do I act when there is extra space available? How will the flex items divvy up the remaining space?

Fig 4 align-items:center; will centre our items along the cross axis, which in our case is top-to-bottom

- `flex-shrink`: How do I act when there isn't enough room for all the flex items? Rather than overflow the container, who will give up part of themselves to make everything fit?
- `flex-basis`: Instead of setting a definite width or a height on your element, ideally what width (as a row) or height (as a column) will it be?

Note that while it is possible to specify these properties individually, you will almost always be using the `flex` shorthand to specify the `grow`, `shrink` and `basis` values all at once. Check the videos at *Flexbox.io* for a more detailed description of the `flex` shorthand property.

Growing and shrinking

The idea is that we can set our ideal width or height with the basis value, and then when there is extra space available for the flex items, the `flex-grow` property will decide how much extra to take up. Similarly, when there is not enough space

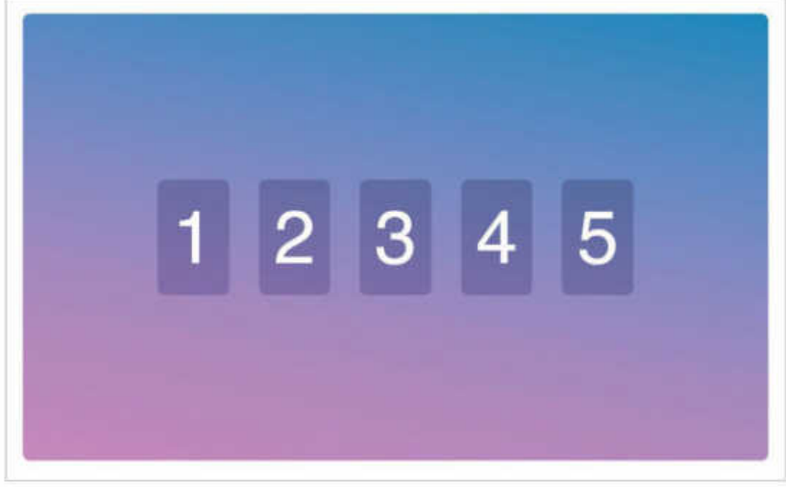
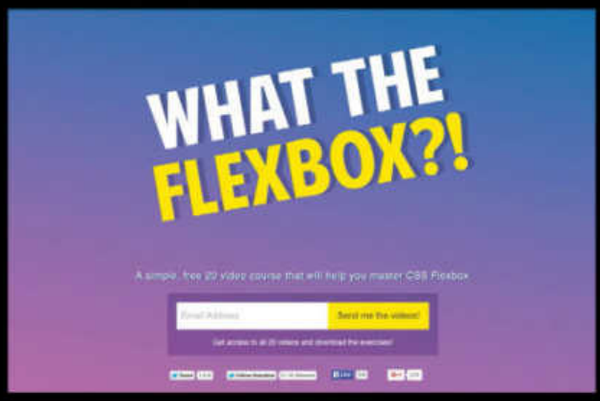


Fig 5 Justify-content:center; will centre our items along the main axis

RESOURCES

MUST-HAVE RESOURCES



What the flexbox?!

flexbox.io

'What the flexbox?!' is a free 20-video training course I've created. In the first half of the course, each video introduces a new aspect of flexbox. I've kept these nice and short, so you can reference them later when you need to brush up on a particular part. The second half of the course dives into a real-world example, detailing how we can use flexbox to quickly and easily solve many of the common layout problems we face.

A complete guide to flexbox

netm.ag/guide-275

Once you get the hang of flexbox, you can put the training materials aside. However, it's helpful to keep a visual reference handy. This fantastic resource from Chris Coyier breaks down each of the 13 different flexbox properties, showing which apply to the flex container and which apply directly to the flex items.

Flexbugs

netm.ag/bugs-275

Flexbox is not without fault, and like anything, there are a handful of cross-browser bugs and workarounds you should know about. Flexbugs outlines a range of known flexbox bugs as well as offering possible fixes and workarounds.

Flexbox in 5 minutes

flexboxin5.com

For a quick'n'easy introduction to the fundamentals of flexbox, check out 'Flexbox in 5 minutes'. It's a step-by-step wizard that will guide you through how to create, style and change your first flexbox-based layout.

available, the `shrink` property will decide how much each element will give up (or 'shrink').

The `flex-grow` and `flex-shrink` properties are unitless, proportional values. They describe how much – in relation to all the other flex items – the item will grow or shrink.

Let's say we have two flex items: video and credits. We will set the video to `flex:1 1 700px;` and the credits to `flex:3 3 300px;`. Now the parent of both of these items is the flex container, and when it is 1000px wide things work out perfectly: the video takes up its 700px and the credits take up the other 300px.

What happens when the flex container is 1500px wide? We have an extra 500px to work with, so where does that go? That is where `flex-grow` kicks in. The video is set to `1` while credits is set to `3`. That means of all the extra room, credits will

It has been a while since anything this large has come to CSS, and it's a tough thing to learn

take three times (375px) the amount of space the video will get (125px).

Similarity, what happens when our flex container is smaller than 1000px? Let's say it's 900px: how do the video and the credits act then? Unlike with floats, we don't just break onto a new line, or scale them down with percentages. Instead, we use the `flex-shrink` property.

Since the credits have a `flex-shrink` of `3` and the video has a `flex-shrink` of `1`, this means the credits will give up three times as much space as the video. So since we need to shave off 100px from somewhere, the credits will give up 75px, while the video container will only give up 25px.

Fig 6 align-items:center; will centre our items along the cross axis, which in our case is top-to-bottom

UNKNOWN NAVIGATION SIZE

Let's look at some common use cases of flexbox. If you have ever worked with a navigation in a CMS like WordPress, you'll know that it can be hard to predict how many elements will be included in your navigation.

Evenly distributing the space between all the elements requires JavaScript. For example if you have three elements, each one should be take up 33.33 per cent of the width, while with five elements each one should take up 20 per cent.

Let's take a look at this commonly seen code as an example:

```
<ul class="nav">
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
</ul>
```

With flexbox, we can easily create this navigation, and even make it responsive, all with just a few lines of CSS. All we need to do is to set our navigation container (usually an unordered list of items) to `display:flex`, and then each of the flex items to `flex:1` or `flex-grow:1;`.

This will stretch the list items horizontally and fit them perfectly into the available width. The reason this works so nicely is that we set the flex items to grow `1`, which means that when there is extra space left over, it will be divvied up evenly between all the items.

For more on this, and to learn how to size your navigation elements differently, make sure to watch the responsive navigation tutorial available on *Flexbox.io*: *wes.io/dWnh*.

EQUAL-HEIGHT COLUMNS

Have you ever wished that CSS had a `height:as-high-as-the-highest-sibling` property? We have all been there – we have three columns of content, all of which are different sizes (fig 6). The content is dynamic and the site is responsive, so setting a fixed height on each one is out of the question, and a JavaScript fix isn't ideal.

Earlier we learned that the default of `align-items` is stretch. This means the flex items will stretch to fit the parent flex container. And how is the height of the flex container defined? Almost always by the height of the tallest content box.

Let's take the following markup, for example. If we render this out with floats and percentage widths, we will see the container is sized by the middle element and the other two are only as high as they need to be.

Fig 7 With flexbox, boxes stretch to fit the flex container and all become equal height

```
<div class="container">
 <div class="box">
  <p>I'm short</p>
 </div>
 <div class="box">
  <p>I'm a pretty tall box that is the biggest</p>
 </div>
 <div class="box">
  <p>I'm a medium sized box</p>
 </div>
</div>
```

Now, if we simply use `display:flex;` on the flex container, and set each element to be 33.33 per cent with the `flex-basis` property, immediately the flex items stretch across the entire cross axis, regardless of how much content is in them (fig 7).

```
.container {
 display:flex;
}

.box {
 flex: 1 1 33.33%;
}
```

CONCLUSION

I hope by now you see the value in learning how to use the Flexible Box module. While it won't solve every issue you have with CSS, it's an important tool every designer and developer should know, and have in their arsenal.

It has been a while since we have had anything this large come to CSS, and I'd argue it's one of the tougher parts of CSS to learn. Just remember that you pushed though learning floats, so flexbox is totally something you can master!

ABOUT THE AUTHOR

RACHEL ANDREW

w: *rachelandrew.co.uk*

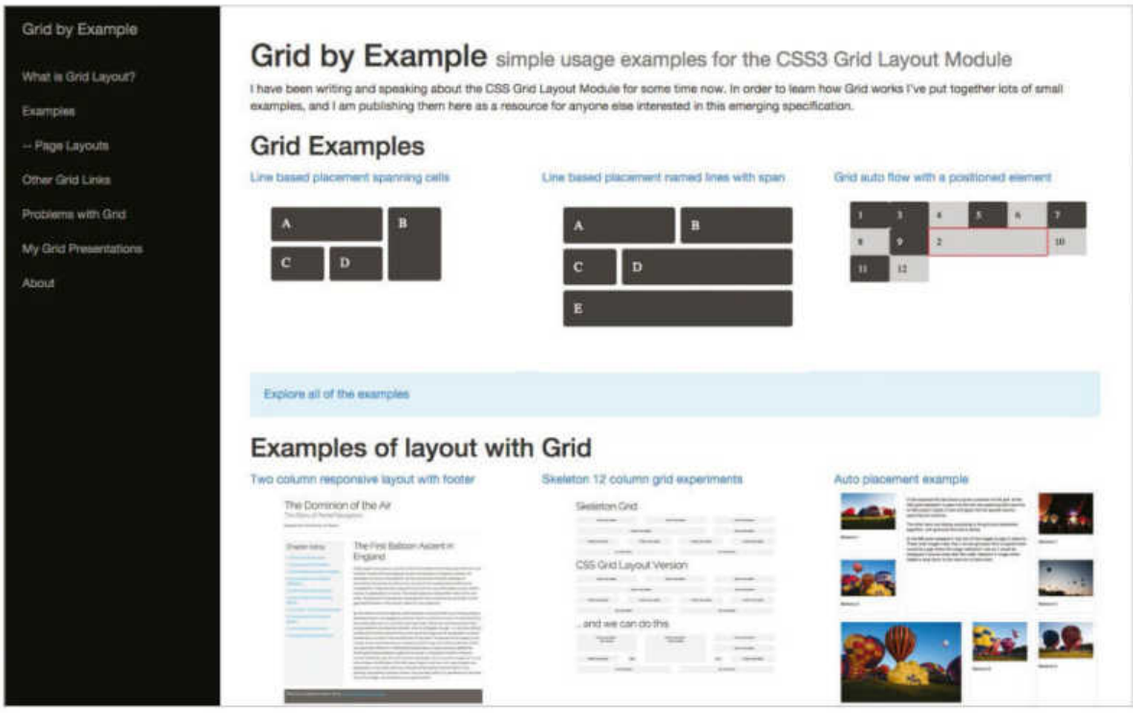
t: @rachelandrew

job: Co-founder, Perch CMS

areas of expertise: Front- and backend web development

q: what's your guiltiest pleasure?

a: I don't have things I like doing that I feel guilty about – I wouldn't do them if that was the case!



CSS

INTRODUCING THE CSS GRID LAYOUT SPEC

Rachel Andrew explains how the upcoming CSS Grid Layout spec will enable designers to tackle two-dimensional grids

For over half of my career on the web, frontend development involved a lot of creating images in Photoshop, then chopping them up and using markup to construct the design in a browser. Thankfully, CSS has evolved to a level at which we don't need to do much of that any more. Creating rounded corners, gradients and even animations are now possible just using CSS.

One area of CSS hasn't evolved quite so quickly: CSS for layout. We are still reliant on fragile floats, positioning and display inline-block or table, and are increasingly coping by using frameworks like Bootstrap. However, things are changing, and one of the most exciting new modules is CSS Grid Layout. In this article I'm going to introduce the module with some simple examples. It's a big and complex specification, but once broken down the individual concepts are pretty straightforward.

WHAT IS CSS GRID LAYOUT?

CSS Grid Layout is a specification for creating two-dimensional grids. It will give you the ability to set up a grid for your webpage and place elements precisely on it. The specification is currently in Working Draft status, and there is an excellent implementation behind a flag in Chrome.

Grid is a companion to the Flexible Box Module (flexbox). Flexbox is designed for one-dimensional layout, so things can be arranged in an unbroken line. Grid is designed for two-dimensional layout, meaning the items don't need to sit next to each other. In the future we're likely to use both: Grid

VIDEO

Rachel Andrew has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/gridvid-272*

Layout for main page areas, and flexbox for the smaller UI elements it excels with.

CREATING A GRID

To create a grid, we need to use the grid or inline-grid value of the display property, and then set up rows and columns. My page has a wrapper element with a class of `grid`. Inside this is a header, nav, sidebar and a footer.

```
<div class="grid">
 <header></header>
 <div class="content"></div>
 <nav></nav>
 <aside></aside>
 <footer></footer>
</div>
```

I then create a grid with CSS:

```
.grid {
 display: grid;
 grid-template-columns: repeat(5, 1fr);
 grid-template-rows: repeat(3, auto);
}
```

The spec will enable you to set up a grid for your webpage and place elements precisely on it

The grid has five equal-width columns and three auto-size rows. The rows will expand to the height of their content. The `repeat` keyword repeats the pattern after the comma the number of times stated before the comma. So our `grid-template-columns` value could also be written:

```
grid-template-columns: 1fr 1fr 1fr 1fr 1fr;
```

All the immediate child elements of `.grid` are now able to be positioned on the grid. There are two main methods of placing items – by line or template area.

Positioning by line numbers

Grids have lines that can be referred to by number or name. Our five-column grid has six lines and four rows, including the line after the final column track and final row track. To position my elements as a standard layout with a header, a main content area with a sidebar on the left and right, and a footer, I would do the following (code at *netm.ag/line-272*):

IN-DEPTH

TRY OUT GRID LAYOUT FOR YOURSELF

At the time of writing Grid is available behind the Experimental Web Platform features flag in Chrome and Opera, and also prefixed in WebKit Nightlies. This implementation is usable and covers the majority of the spec. It is hoped that at some point this year Grid will be shipped in browsers that use Blink – keep an eye on *gridbyexample.com/browsers* for full and updated browser information.

Chrome and Opera

To see the examples here, or any other Grid Layout examples in action, you will need to enable the Experimental Web Platform Features flag in Chrome or Opera. Both of these browsers use the Blink rendering engine, which has the most complete and up-to-date implementation of Grid.

In Chrome, type *chrome://flags/#enable-experimental-web-platform-features* into your address bar. In Opera use *opera://flags/#enable-experimental-web-platform-features*. You can then enable that flag and restart your browser. You will be able to take a look at the examples and try out your own experiments with Grid. Blink does not require a prefix for Grid to work.

WebKit

You can try out Grid in WebKit by downloading a WebKit nightly build from *nightly.webkit.org*. You will need to use a `-webkit` prefix.

Gecko

Grid Layout is currently being implemented in Gecko. You can try it out in Firefox by typing *about:config* in your address bar and enabling the flag 'layout.css.grid.enabled'.

Internet Explorer

What about Internet Explorer? You might be interested to know that there has been an implementation of CSS Grid Layout in Internet Explorer since IE10. The specification started out in Internet Explorer, so the implementation that you will find in IE10 and IE11 is based on that very early spec. Hopefully a revision of that implementation will happen soon so we can see the latest specification in all browsers.

If you are interested in that older implementation, take a look at a piece I wrote for 24 Ways in 2012 at *netm.ag/24-272*, which includes links to IE examples.

FOCUS ON

GRID TERMINOLOGY

CSS Grid Layout introduces a whole set of new CSS properties and comes with its own terminology.

Grid Lines
In this article we look at Grid Lines – these can be horizontal or vertical, addressed by a number or name.

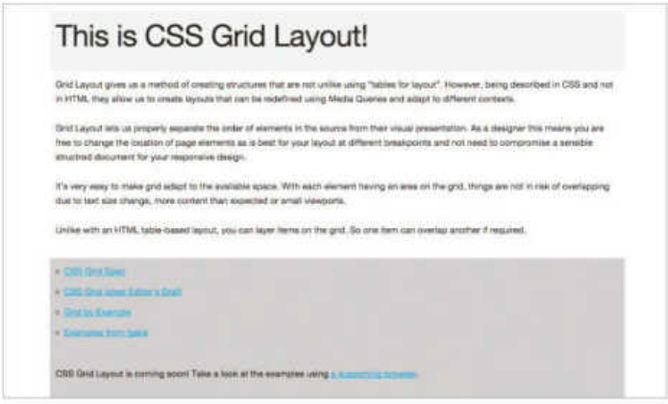
Grid Tracks
A track is any single row or column between two Grid Lines. When we place an element into a single row or column, it is going into a single track on the Grid.

Grid Cells
A Grid Cell is the smallest unit on our grid, and is surrounded by four Grid Lines. Conceptually it is just like a table cell.

Grid Areas
A Grid Area is a collection of Grid Cells. The boundaries are defined by any four Grid Lines intersecting. When we use line-based positioning we define the Grid Area by specifying start and end lines, whereas when we use Grid Template Areas we define the area by repeating the name of the area.

Those readers who were unlucky enough to have to build layouts using tables might feel Grid is somewhat similar, and in some conceptual ways it is. However, Grid has a couple of obvious advantages over tables.

Firstly, it does not tie your layout to the document structure as is very much the case in tables – there is no indication of what the layout should be in the markup. Secondly we can redefine the layout using CSS, according to any breakpoints we choose.



Starting point Our layout before creating a grid

```
header {
 grid-column: 1 / 6; grid-row: 1;
}
.content {
 grid-column: 2 / 5; grid-row: 2;
}
nav {
 grid-column: 1 / 2 ; grid-row: 2;
}
aside {
 grid-column: 5 ; grid-row: 2;
}
footer {
 grid-column: 1 / 6; grid-row: 3;
}
```

The `grid-column` property is a shorthand for `grid-column-start` and `grid-column end`. The value before the `/` is the value for start, the value after for end. The `grid-row` property is shorthand for `grid-row-start` and `grid-row-end`.

So we specify the line on which the content starts and ends. Our header for example starts on the first line and finishes on the last line. If content only spans one track, you can omit the end value, as spanning a single track is the default.

It is worth noting that by default the columns will stretch to full height – you can see this from the grey background on the sidebars. No clearing is needed. The footer, for example, is sat in its own row. It can't jump up and cover content above, no matter which column is the longest.

Positioning by template areas

The second method of positioning items involves creating template areas and placing items onto them (*netm.ag/template-272*). If we have the same markup, we first need to assign a name to the elements we want to position using the `grid-area` property.

```
header {
 grid-area: header;
}
.content {
 grid-area: content;
}
nav {
 grid-area: nav;
}
aside {
 grid-area: sidebar;
}
footer {
 grid-area: footer;
}
```

With each area named I use the `grid-template-areas` property on the parent element – in our case `.grid`.

```
.grid {
 display: grid;
 grid-template-columns: repeat(5, 1fr);
 grid-template-rows: repeat(3, auto);
 grid-template-areas:
"header header  header  header  header "
"nav    content content content sidebar"
"footer footer  footer  footer  footer";
}
```

The value of `grid-template-areas` uses the names we assigned to each element to position them. If a name is repeated, it means we want the area to span across that track. Here, our header and footer span all the columns, but the nav is in a single column to leave space for the content, which spans three.

We don't have to define the layout using classes, so we can add lots of small breakpoints

To leave a column empty, you use a full stop character (`.`). So if I wanted the footer to just sit under the content and not the sidebars, I could describe my layout as follows:

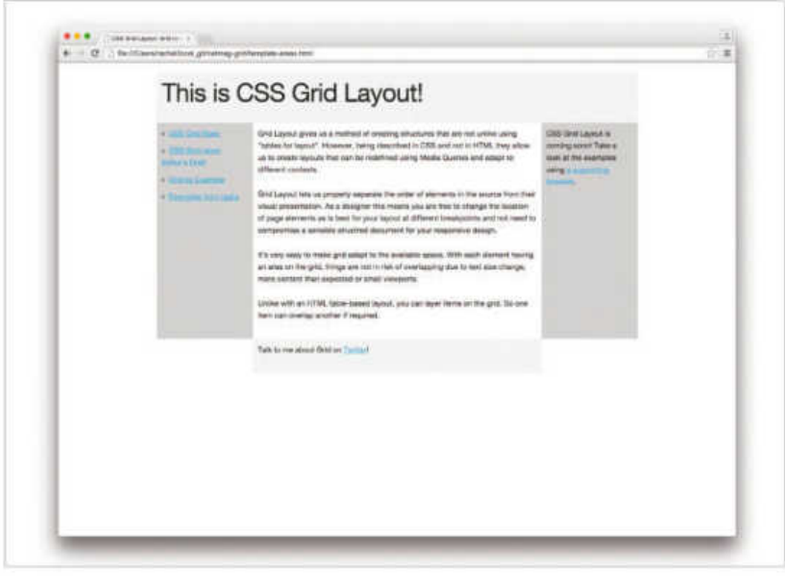
```
grid-template-areas:
 "header header header header header "
 "nav content content content sidebar"
 ". footer footer footer  . ";
```

The spec allows the use of one or more consecutive `.`s to indicate a null cell. This can help you to line up your ascii art to make it easier to read.

GRID AND RWD

CSS Grid Layout means you can redefine your grid or the position of things on it at any breakpoint. Because Grid can display things out of source order, you can order your document in the most accessible way, moving things around for each breakpoint.

With our template areas example, we might choose to only go to the three-column layout for wider screens, and a single-column layout for mobile devices. Outside of any media queries, I would name my elements as before and then lay out a single column, using Grid to place items in the order I think is best for my mobile users.



Laid out The layout after positioning the items. Here, the footer is set to only span under the content

```
.grid {
 display: grid;
 grid-template-columns: repeat(5, 1fr);
 grid-template-rows: repeat(3, auto);
 grid-template-areas:
"header header header header header"
"nav nav nav nav nav"
"content content content content content"
"sidebar sidebar sidebar sidebar sidebar"
"footer footer footer footer footer";
}
```

Then inside media queries at my chosen breakpoint, I'd redefine that layout back to three columns.

```
@media (min-width: 550px) {
 .grid {
  grid-template-areas:
"header header  header  header  header "
"nav    content content content sidebar"
"footer footer  footer  footer  footer";
 }
}
```

As we don't have to define the layout by adding classes, we can add as many small breakpoints as needed for different devices.

FUTURE TECHNOLOGY

The CSS Working Group is keen to hear feedback as it finalises the spec – you'll find the latest draft at *netm.ag/module-272*. I'm also building a collection of examples you can use as starting points for your experiments (*gridbyexample.com*). Now is a great time to be part of the exciting future of CSS layout.

gTALK

Rachel Andrew explored the CSS Grid Layout spec at Generate London 2015. Watch her talk at *netm.ag/grid-278*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/querypen-275*

ABOUT THE AUTHOR

PHILIP ZASTROW

w: *zastrow.co*

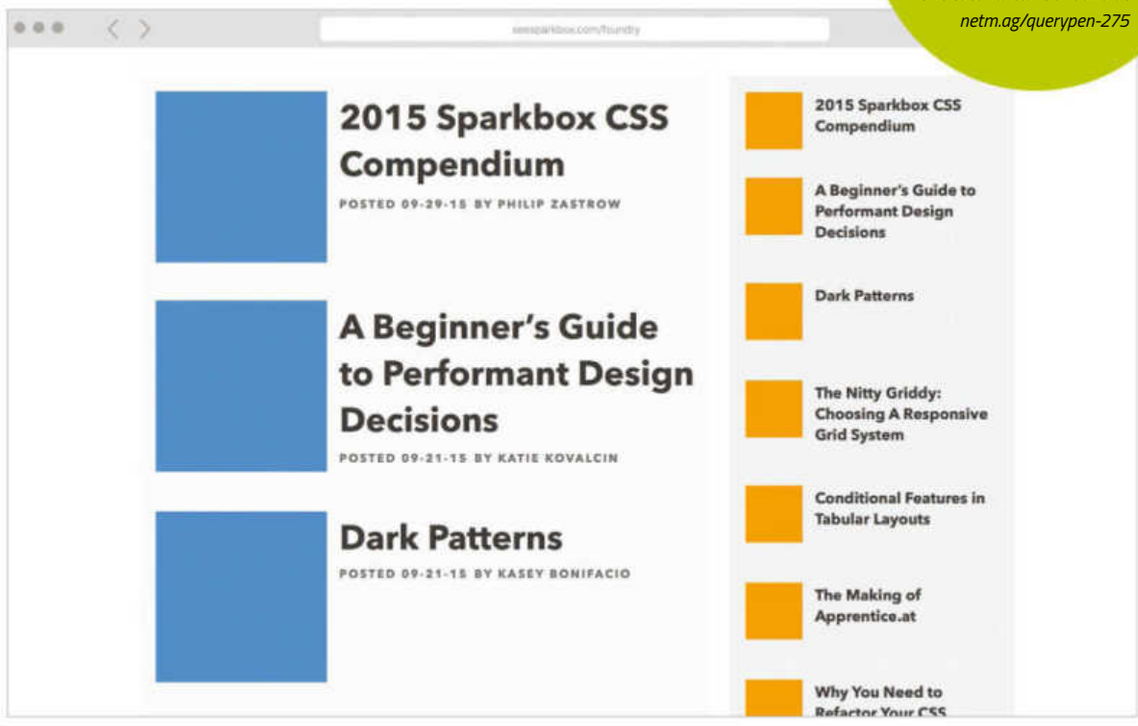
t: @zastrow

job: Developer, Sparkbox

areas of expertise: Frontend design and development

q: what's the first album you ever bought?

a: The one that stands out is a jazz collection from the saxophonist Charlie Parker. I was really into jazz in my teens



ELEMENT QUERIES

CONTAINER-BASED RWD WITH ELEMENT QUERIES

Philip Zastrow examines how element queries will change the way we create design systems and approach reusable components

When RWD was invented, it completely changed how we thought about design and approached problems. Now, element queries are about to have a similar impact on how we approach design patterns, encouraging the creation of design systems. Element queries are RWD levelling up.

MEDIA QUERY LIMITATIONS

Information about the browser window or device screen size is the most reliable way to determine changes to a website design. Although we can achieve quite a lot with media queries, we are unable to make design decisions for individual components based on anything other than the size of the viewport. A component is locked to a rigid set of requirements. Changes happen through media queries of the screen size, yet a component's location within the layout is an unknown variable. The results of these queries will modify the layout of the entire page, not the component alone.

For example, let's say we have a class of `.image-container` that makes up 20 per cent width on small screens. When its parent is 600 pixels or larger, `.image-container` should jump up to 35 per cent width. To make this scenario work, we set `.image-container` to `width: 20%;` and write a media query for when the parent reaches a width of 600px.

This won't take effect at a viewport width of 600px – the parent won't be that size until the viewport is at least 900px wide. The media query to target the width of the parent must be written with that viewport in mind.

RESOURCE

Bearded's Patrick Fulton gave a talk entitled 'Element (container) queries' at BDConf DC. Take a look at his slide deck at *netm.ag/fulton-275*

```
@media (min-width: 900px) {
  .image-container {
    width: 35%;
  }
}
```

Let's look at another scenario with that same media query. A 300px sidebar sits next to the parent of `.image-container`. What happens if we need to reuse the `.image-container` in that sidebar at a 900px viewport size?

The container gains 35 per cent width. However, as its parent is less than our 600px width, it would be better for the container to gain 20 per cent width. In order to modify the width at the correct media query, the CSS needs to be adjusted to include the new parent. Instead of the component size being conditional on its width, it is dependent on its parent's name and the size of the viewport.

Elements can adapt to the constraints of the layout, independent from the viewport

WHAT ARE ELEMENT QUERIES?

Problems like this are persistent when creating responsive sites. Efforts to componentise websites, like Pattern Lab, have latent potential. However, elements still need to be designed within the scope of a page layout. The next step in making RWD more fluid is to enable any site component to live anywhere in the page layout at any time.

Element queries are an attempt to bring responsive design to a component level, by setting breakpoints based on the conditions of elements instead of the viewport. This allows greater, granular control of an element, so it is capable of effectively adapting to the constraints of the layout, independent from the viewport shape.

Element Query polyfills

Since element queries are still in an early draft, we have to rely on JavaScript polyfill implementations for now. There are a handful of polyfill tools to choose from, and many will support browsers as far back as IE7 (which is better supported than even media queries). In this article, I'll be diving into eq.js by Sam Richard: *github.com/Snugug/eq.js*.

Let's say we are running a news site. With each article we have the same information: a photo, a headline, the date and the author's name. The article

IN-DEPTH

CONTAINER QUERIES

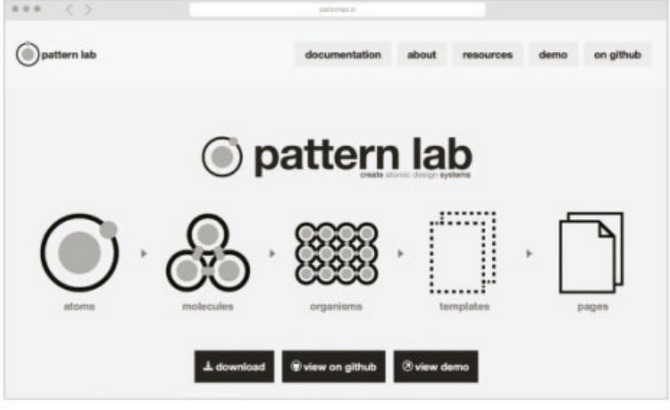
The RICG has carried the torch on element queries (*netm.ag/usecase-275*) for a while, and prompted an upswing in conversation surrounding the proposed feature in the early half of this year. This, however, has led to the inevitable realisation that element queries, as proposed, present one fundamental issue: an infinite loop of style changes.

```
.article:media(min-width: 600px) {
  width: 599px;
}
```

You may be thinking, here I've written an elaborate explanation of element queries today, and now I'm saying it's irrelevant? Not at all. Element query polyfills are still quite usable and viable today. What has happened is a slight shift in the conversation around making element queries a web standard.

The result is to change the rules around how an element query can be implemented. Instead of allowing an element to query itself, it must now rely on its containing element. Mat Marquis wrote at length about this shift at *netm.ag/marquis-274*. Fundamentally, this affects very few of the use cases for element queries; even the name isn't in any real need of a change, as elements are still being queried.

When implementing element query polyfills keep this approach in mind, as there is a very real possibility of initiating an infinitely looping style block. Much of this tutorial follows a container query method; scoping styles to parent element queries instead of the elements themselves. Most of the polyfills do not account for this possibility, so use good judgement and test often when implementing element queries.



Component toolkit Pattern Lab by Brad Frost and Dave Olsen is a great way to utilise reusable components in a website

FOCUS ON

RESOURCES

Since Element Queries are still in an early draft, we have to rely on JavaScript polyfill implementations. There are a handful to choose from:

- ElementQuery (*github.com/tysonmatanich/elementQuery*)
- CSS Element Queries (*https://github.com/Snugug/eq.js*)
- EQCSS (*eqcss.github.io/eqcss*)
- eq.js (*github.com/Snugug/eq.js*)

Take a look at these articles and websites for further reading on element queries, their use cases, and even how to get more involved in making element queries a web standard.

- Responsive Issues Community Group (*ricg.io*)
- Media queries are not the answer (*netm.ag/matanich-275*)
- Working around a lack of element queries (*netm.ag/jehl-275*)
- Element queries, from the feet up (*netm.ag/buchner-275*)
- Media queries are a hack (*netm.ag/taylor-275*)
- Element queries for CSS (*netm.ag/hodgins-275*)
- Element queries (*xanthir.com/b4PR0*)

Most element query polyfills rely on attribute selectors to accomplish styling at various widths. Attribute selectors are a unique and powerful way to target styles on an element. Take a look at 'The skinny on CSS Attribute Selectors' *(netm.ag/coyier-275)* by Chris Coyier to discover how to master their potential.

Large screen view On larger screens the components are kept small with CSS columns, only displaying the article title

component can show up in any of three places: the main content area (`<main>`); a supplemental content area (`<aside>`), which is below the main content on small screens and in a smaller right-hand column on larger screens; and finally the footer (`<footer>`).

Each area has decreasing size constraints on the article component. `main` allows the component to grow to its largest possible size. `aside` keeps the component at a middling size. Whenever the `aside` is as wide as the `main` , the contents of the `aside` is broken into two columns. Meanwhile the `footer` will always remain in its smallest format. This will be enforced by increasing the number of columns using media queries.

To begin we need to define our component:

```
<article class="article">
  <header class="article-header">
    <h1 class="title">...</h1>
    <p class="author">...</p>
    <p class="date">...</p>
  </header>
  <div class="image">
    <img src="{{article.link}}">
  </div>
</article>
```

Data attributes and JavaScript

With eq.js we need to define our widths within our HTML with a data attribute. The JavaScript watches the width of the element with this data attribute, and when one of the specified widths is hit, it creates and updates another data attribute that is used in the CSS to modify the styles.

For our news items we have some very specific sizing events to meet, as well as some content priorities to consider. The most important element is the headline, followed by the date, the image, and finally the author name.

With this in mind, we can make informed design decisions based on the space available.

```
<article ... data-eq-pts="small: 300, medium: 400, large: 600">
```

Working with the element alone on a page, we have been able to define the necessary breakpoints as 300px, 400px and 600px. Below 300px only the article title is displayed. At 300px we bring in the article's post date. At 400px we bring in the article image and author name. At 600px we can increase the size of the image.

eq.js watches the width of each `.article` on the page, and will add `data-eq-state="small"` to the element when the first breakpoint is reached. As the element

continues to grow, each additional key name will be added to a space-separated list in the attribute.

By the time `.article` hits a minimum width of 600px, the attribute will contain all three key names: `data-eq-state="small medium large"`. These dynamic attribute values, paired with attribute selectors in our CSS, are the tools needed for our element queries.

Attribute selectors

We can scope our CSS to specific widths only by using an attribute selector with a dollar sign: `.article[data-eq-state$="medium"] {...}`. Any styles applied in this manner will only display when the attribute value ends with `medium`.

We can also set more min-width-like styling using the tilde: `.article[data-eq-state~="medium"] {...}`. This type of attribute selector will apply styles wherever medium is within the `data-eq-state` attribute (here, the medium and large breakpoints).

We want to start off by hiding the article's date, image, and author at the smallest view of the module. As these are supplemental content to our article title, hiding these items does not diminish our content. To keep this simple we will just add

Element queries afford us the ability to add contextual design if the space is available

a `display: none;` to `.author`, `.date` and `.image`, and bring them in with attribute selectors.

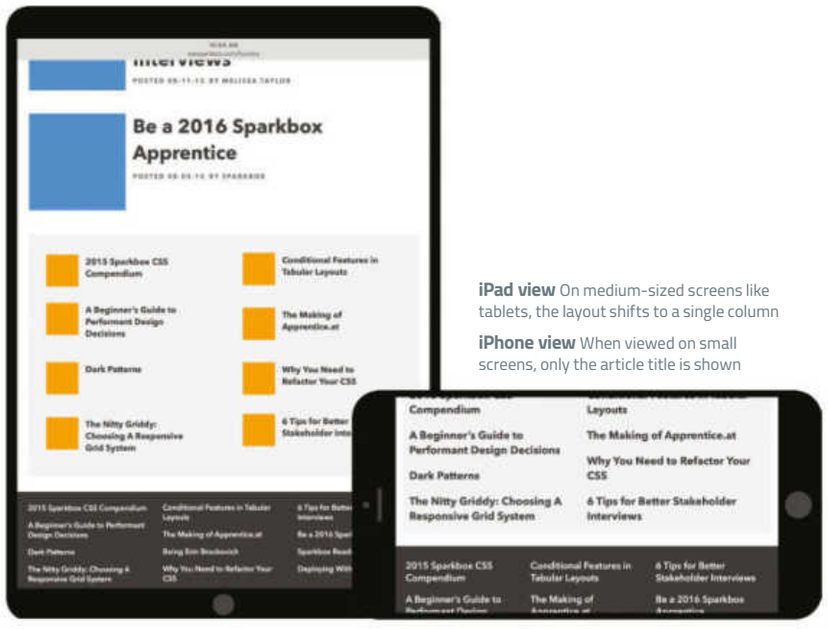
For our possible layouts and uses, this means our module will only display the article title when the module is less than 300px wide. Our first breakpoint adds in the post date of the article.

```
.article[data-eq-state~="small"] .date {
  display: block;
}
```

At 400px wide we will pull in the article image and the author.

```
.article[data-eq-state~="medium"] .author,
.article[data-eq-state~="medium"] .image {
 display: block.
}
```

Then, of course, we want to make sure our image makes the jump from 20 per cent to 35 per cent when the parent reaches a width of 600px.



iPad view On medium-sized screens like tablets, the layout shifts to a single column

iPhone view When viewed on small screens, only the article title is shown

```
.article[data-eq-state$="large"] .image {
  width: 35%;
}
```

Now wherever this reusable component is placed on the page, it will format the content and the design to fit that space.

Contextual design

One of the wonderful things that element queries affords us is the ability to add contextual design if the space is available in an element. Furthermore, this can be an element query in and of itself.

There are two pieces of content within our component that could use some additional context when the space allows. The first is the author. It is nice to have the author listed, but, we can add 'By' or 'Written by'. A similar extra level of contextual information can enhance the date. A 'Posted on' can be added, or a shorthand date could be extended to a full-length version.

These types of enhancements are unpredictable with media queries. However, element queries give a new level of precision to not only design, but also contextual content.

FUTURE OF ELEMENT QUERIES

There is still a long road ahead for the element query. Thankfully the Responsive Issues Community Group (RICG) has taken the lead to see element queries in some form or another become a web standard.

Native implementation aside, the abilities of element queries, and reasons for making the shift to using them today, are vast. The possibilities of what element queries can bring to the future of web design are quite exciting.

RESOURCE

Author Philip Zastrow gave a talk entitled 'Container-based RWD with element queries' at CSS Dev Conf. You can take a look at the slide deck at *netm.ag/deck-275*

View source files here!

Download the assets you need for this tutorial at *netm.ag/pixatepics-265*

ABOUT THE AUTHOR

JULIAN SHAPIRO

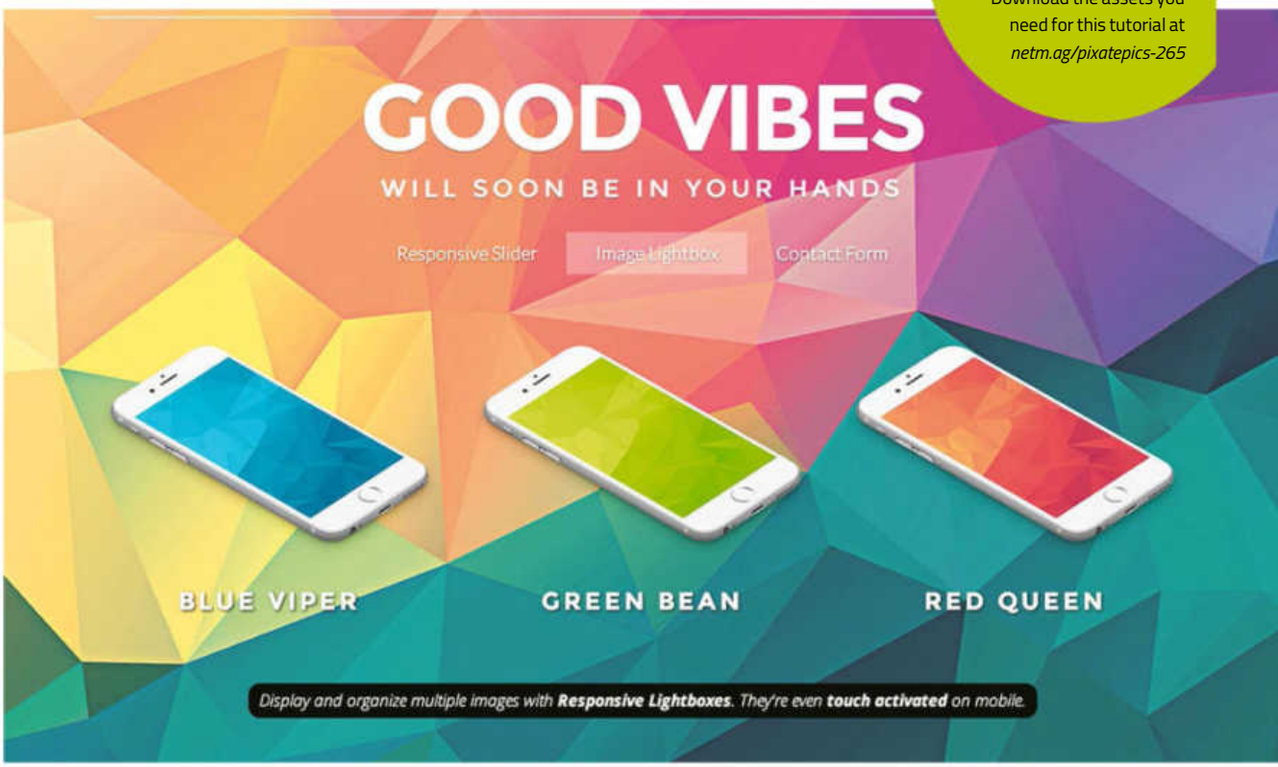
w: *julian.com*

t: @Shapiro

job: Web designer

areas of expertise:
HTML, CSS, JavaScript

q: what's the worst holiday you've ever been on?
a: Any time I get suckered into bowling



WEBFLOW

BUILD WORKING SITES QUICKLY WITH WEBFLOW

Julian Shapiro explains how Webflow enables designers to build feature-rich, production-ready sites in no time at all

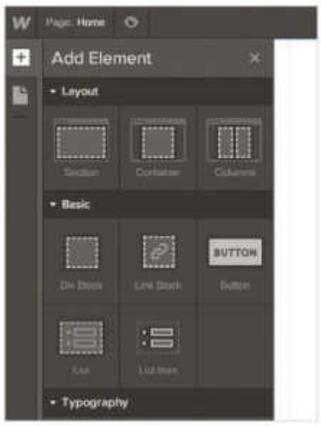
Many site-building tools take a developer-first approach, making it difficult for designers to work with them. Thankfully, Webflow (*webflow.com*) has corrected that mistake, providing designers with the Photoshop-like pixel precision they need to build a production-ready website, without the assistance of a developer. Webflow is not merely a toy for those who don't know how to code; it allows you to build sites from scratch, giving you the power to precisely meet client specifications. What's more, Webflow is now offering a free plan – perfect for getting started and exploring what the tool has to offer.

In this tutorial, I'll show you how to use Webflow to design a page that showcases rich multimedia content. In doing so, you'll witness the enormous workflow benefits that using a site-builder can bring. We'll create an interactive display of content using a slider widget, along with a lightbox that showcases retina-quality images. I'll show you how to make sure both of these components look as good on mobile as they do on the desktop – without fiddling around with CSS media queries. We'll also create a navigation bar that responsively transforms into a drop-down menu on smaller screens.

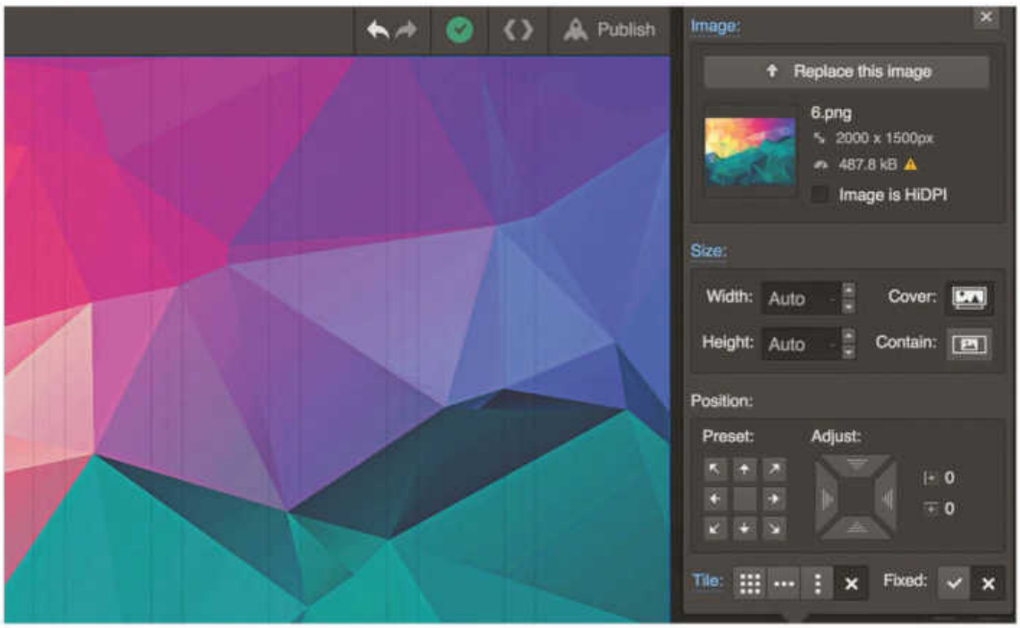
We'll then finish it off with a form that will enable you to collect emails from your visitors so you can stay in touch with them. And we'll do all that – and make our design fully responsive – in just 16 steps.

VIDEO

Julian Shapiro has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/webflowvid-266*



Step 1 Begin building your layout using Sections, Containers and Columns



Step 2 Elements can be filled with images, colours and gradients

01 Webflow's site builder starts you off with a blank canvas set on a 940px grid. You can switch view modes between 'Desktop', 'Tablet', 'Phone-Landscape' and 'Phone-Portrait' by toggling the device icons in the top toolbar. Webflow's workflow is optimised so changes cascade downwards. I'll first design the site in desktop mode, then optimise for smaller screens. Let's begin by opening the `Add Element` panel (keyboard shortcut `A`) and dragging a 'Section' (the equivalent of a content wrapper) onto our canvas.

02 Let's spice up this blank canvas. To edit the styling of an element, a 'class' must be created in the `Style` panel (note that you can also add secondary classes, to help differentiate styles across related elements).

Inside the `Style` panel (`S`), you'll find the `Background` palette. From here, let's fill the body with a vibrant background image. Using the `Cover` style will guarantee that your image 'covers' the full width, while the `Fixed` position ensures it scrolls with the rest of the site.

03 You want visitors to be able to efficiently navigate your site – especially on mobile devices where screen real estate is limited. A responsive 'Navbar' will do the trick. Creating one in Webflow is easy: From the `Add Elements` panel, drag a Navbar above the Section element we previously added. Position it with the help of the Navigator tool (`F`) – just like you would with a layer in Photoshop. Note the blue lines that appear when moving an element – these are positioning guides, use them to help you accurately place elements.

Step 5 Typographic elements already have default classes

04 In Webflow, you have total control over the styling of any element. With the safety net of undo (`cmd+Z` or the arrow icons), you can confidently play with positioning, gradients, borders and shadows, and you can animate the transition of all these properties. Check out the `Effects` palette to learn more. With your Navbar finished, convert it into a 'Symbol' by clicking the icon in the `Assets` panel (`H`). Symbols allow you to duplicate content into a group that you can thereafter edit in one shot. Remember symbols from Flash? Same thing.

EXPERT TIP

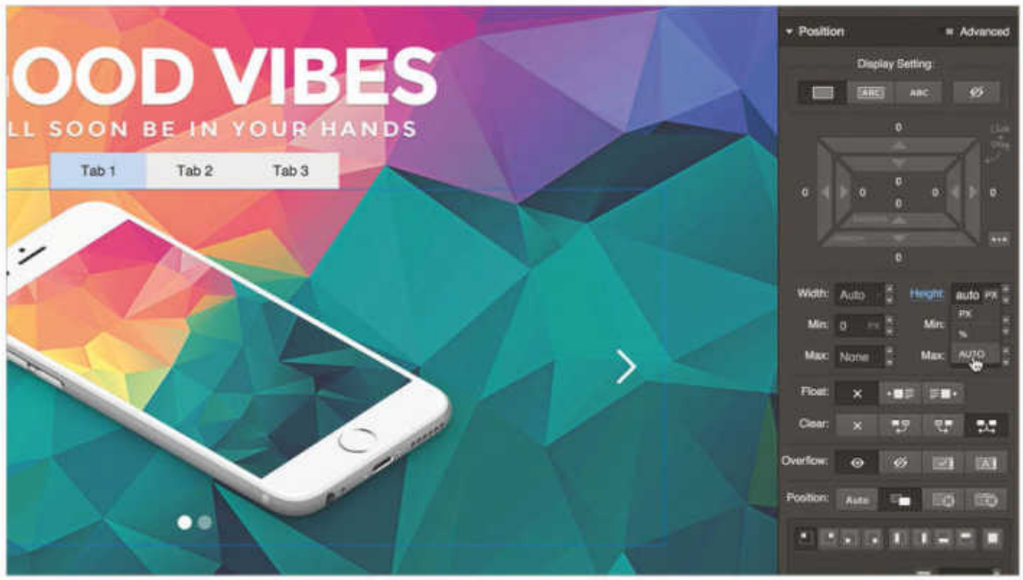
EDITING SHORTCUTS

Webflow has a really awesome shortcut for cutting down time spent completing routine tasks: holding `Alt` while click-dragging on the 'Margin' or 'Padding' buttons enables you to simultaneously edit the opposite value. This saves you from repeatedly copy-pasting your values across different axes.

05 To keep your content from spilling off the sides on narrow screens, add a 'Container' into the Section. Drag and drop a 'Heading' into the Container, aligning it as centred text using the Typography palette. Notice that typography elements already have default classes. Accordingly, when styling a default heading, it will automatically change the default style for all headings, unless you remove the Heading class or add a secondary style class. We now have the essential building blocks for our site: Section, Container, Navbar and Heading.

06 Now the plot thickens. Let's build a slideshow nested inside of a 'Tab' Container. Drag a Tabs element from the widgets library. Tabs are great for organising lots of content and delivering it to the viewer through a single click. You can easily switch between the individual Tabs with the Navigator (F) to individually edit their contents. Also note that, to help find or select a specific element or layer, the Breadcrumbs feature on the bottom toolbar makes selecting parent elements a total breeze.

07 Visitors must be able to interact with your site. Let's give them a slideshow that works responsively across all devices.



Step 7 Create a completely unique slider with animated interactions

Desktop users will be able to click to toggle through the slides, and mobile users will be able to swipe. Let's drag a 'Slider' into the Tab panel for our first Tab. With the auto-play feature, your slider will greet visitors with automatically scrolling content.

08 Let's create an image gallery for the second Tab using the 'Lightbox' widget. A lightbox displays your images in a fullscreen window. You can make the images align in a responsive grid by using a Columns section then placing a Lightbox element inside each column. In the Lightbox Settings palette from Image Settings (D), click the 'Add Image' icon. You can enable users to view multiple images as a group inside the window by toggling 'Link with other Lightboxes' and creating a group name for all your images.

09 Tab three will be our container for the contact form, where visitors can submit their emails. Utilising the same layout controls from the columns, you can make the contact form responsive so mobile users can easily fill out the fields. However, in order to separate the fields into columns, the Columns section must be inside a Form block. After doing that, drag and drop each field into the individual columns. This way, the columns with fields inside can be adjusted for different devices.

10 Why not make it easy for your visitors to share your site with friends? Let's add a new Container below the Tabs. From the widgets library, add the social buttons for Twitter, Google and Facebook. In the Settings (D) for each button, you can edit the 'Share URL' and customise the layout style to display the 'boxes' in a vertical manner. To save screen

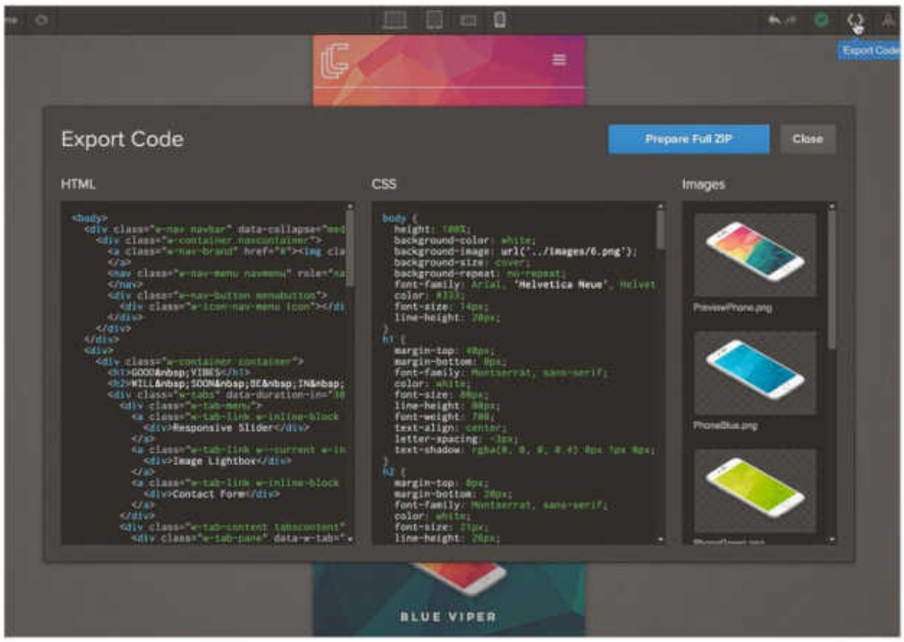
Step 10 Add social sharing buttons to enable visitors to share your content

EXPERT TIP

SAVING VERSIONS

One of the best features Webflow offers in terms of boosting productivity is the ability to create a duplicate version of your site. It gives you the flexibility to completely overhaul your site, without losing what you had before. If you need to restore your site to a previous version, you'll find it stored under 'Backups' in your site's Settings. Webflow automatically saves your progress and creates restore points every 30 minutes, so you don't have to worry about running out of undos.

Step 15 Go live with the push of a button



Step 16 If you're building a more complex website, you can also export your code and build on top of it in your IDE of choice

real estate, let's align them next to each other horizontally by changing the display settings for each button to `Inline-Block`.

11 Now let's preview the site. Click the 'eyeball' icon in the upper-left corner to toggle the preview mode and inspect your site as a live mockup. While previewing, you can also switch between the different view modes to see how your design flows on smaller or larger viewports. Webflow provides four default view modes, but you can test the design's flow at any size by manually shrinking the width of your browser window. This is one of the many benefits of working directly in the browser!

12 We need to bring some more life to this page. Let's add hover states to these buttons so that feel interactive. A site that responds when the user interacts with it is key for establishing a user experience where each element's purpose is clearly understood. In the `Style` panel, above the `Classes` field, you'll find the `States` drop-down. Select `Hover` and create a unique style for your button. With a new style, you can animate its transition. In the `Effects` palette at the bottom of the `Style` panel, add transitions to the style's properties.

13 Let's switch to the 'Tablet' view mode so we can optimise our design for that type of device. Notice that the Navbar button is now an icon indicating a drop-down menu – this means it's responding to the new device's size. Users can now press the icon to open the menu.

14 To design this, open the menu from `Settings` (`D`) and edit your typography as desired. Then switch to 'Phone-Landscape' mode and repeat the style touch-up process. Notice how the contact form automatically adjusts into vertical formation here, thanks to its responsiveness.

Finally, let's switch to 'Phone-Portrait' mode so we can see how things look one size further down. Here, the menu tabs are displayed vertically, as are the lightbox thumbnails and contact form. Pretty neat, no?

15 After optimising your design for the various viewport types and cleaning up any unused styles with the Style Manager (`G`), go ahead and publish your site by clicking the 'rocket' icon in the top-right of the toolbar. With a single click, your site will be live, and you will be provided with the shareable URL. You can still continue to make changes to your site as needed – plus, you can always unpublish your site, or keep it private with a Password.

16 We're on our last step! In the Webflow designer, the export code button in the top toolbar will zip the site's content so you can send it to your client or another developer. To publish the site to your own domain, click the 'Webflow' icon in the top-left corner to go to `Settings`. From there, you can connect your custom domain under the `Hosting` tab.

EXPERT TIP

COLLECT EMAIL ADDRESSES

Acquiring an initial user base is key for any project. A great way of doing this is by collecting visitors' email addresses. Normally, this would involve delicately copy-pasting code into exact locations, before tackling a lot of custom tweaking. In Webflow, you can collect emails with a pre-built form widget. Webflow will save all the form data, then export it as a CSV file for you to use with other programs – all without any code.

ABOUT THE AUTHOR

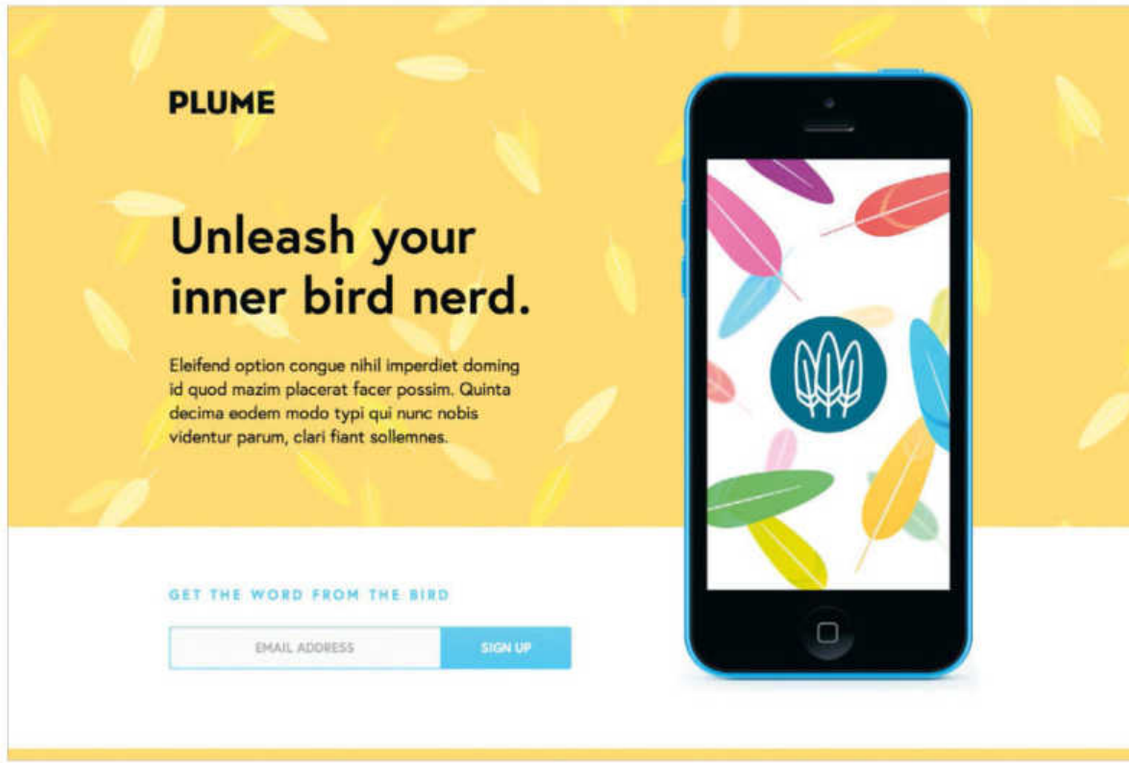
TOM GIANNATTASIO

w: *macaw.co*

t: @attasi

areas of expertise:
Interaction design, CSS3, JavaScript

q: which living person do you most despise?
a: Justin Bieber and King Joffrey



MACAW

CODE-FREE RESPONSIVE DESIGN WITH MACAW

Macaw is a new breed of web design tool – **Tom Giannattasio** explains how it can boost your responsive design workflow

Responsive design is time-consuming and difficult, but Macaw can help alleviate some of those woes. It was built specifically for the modern web, with speed and ease of use in mind. Designers familiar with Photoshop or Fireworks should feel right at home within its interface.

Macaw enbles you to draw and manipulate elements on a canvas. The usual suspects are all present: transforming, drag and drop, nudging, and a new friend called pudge. Elements can be grouped and depths can be managed as if they were layers.

The difference is that all of these familiar manipulations are being translated into HTML and CSS on the fly. When you move an element around the canvas, Macaw automatically calculates the margins, floats, clears and other properties that are necessary to place that element in a static document flow.

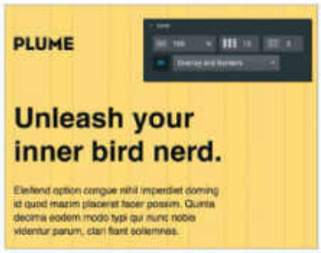
Macaw is built on the same framework as Google's Chrome browser and therefore inherits its rendering capabilities. This opens a wealth of opportunities, especially when it comes to responsive design. You can simply resize the canvas, insert a breakpoint and optimise the layout for different screen sizes, all within a single file.

In this tutorial, I'll show you how to use Macaw to create a responsive coming soon page without touching Photoshop or a single line of code.

RESOURCE

MACAW DOC

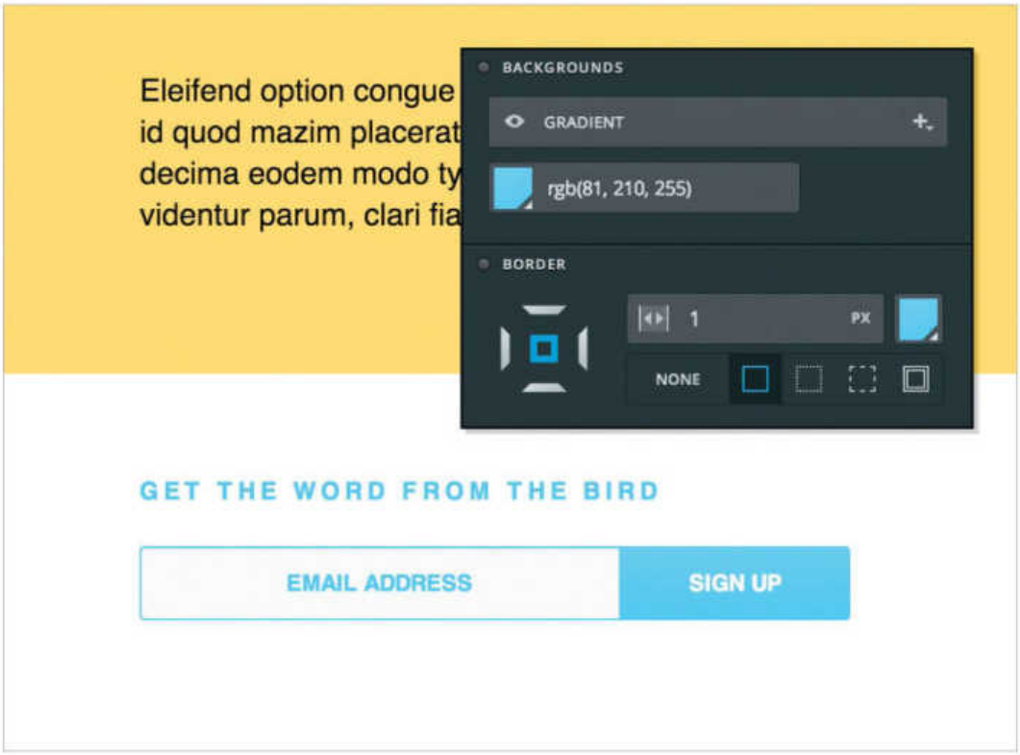
This provides an in-depth description of Macaw's features, and is scattered with expert tips on improving your speed and workflow: *docs.macaw.co*



01 The canvas in Macaw can be resized at any time to see how your design flows in different viewports. It also has a fluid, column-based grid, which can be modified to your liking. The grid provides useful aids during the design process. For example, click Cmd+right or left to nudge an element perfectly to a grid column.

For this project, we'll use a 12-column setup, with a 100% width grid and 3% gutters. You can set these properties in the Grid palette, which is visible when no elements are selected.

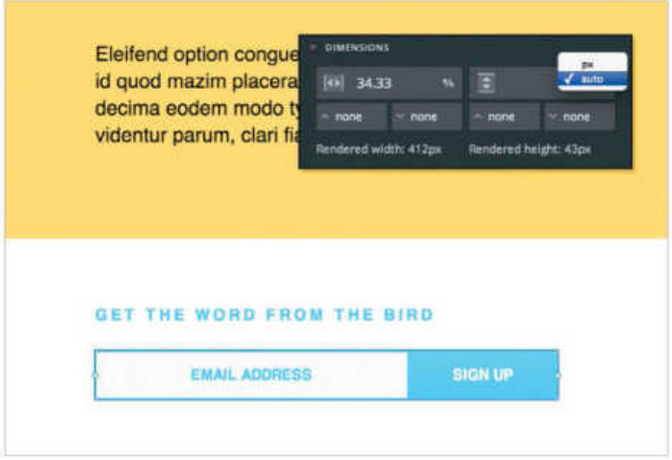
02 Adding elements is as simple as drawing shapes in Photoshop. Select the Input tool (N) and draw a shape on the canvas below the orange header. You'll notice a blinking cursor inside the shape, which indicates that this is a text editable element. Type the words 'Email address' and press Cmd+Return to commit. This will be used as the placeholder text when published.



Styling up It's simple to edit colours and border properties, and add gradients or shadows in the Backgrounds palette

Drag the input into place below the blue header. If your grid is visible, you should be able to snap the input into place. Note the blue guides that appear when moving an element. These are positioning guides, used to help visualise an element's margins and coordinates.

Draw a button next to the email input using the Button tool (B). Again, you can directly edit the text used in the button. Type 'Sign up' and press Cmd+Return to commit.



Creating containers Containers are similar to Photoshop groups, but can receive styling

03 With our elements in place, let's add some styling and advanced options. Select the input using the Select tool (V). The Inspector will update with applicable options. In the section labelled Input Options is a drop-down that lets you set the input's type. Change it to Email .

Now select the button. In the Backgrounds palette, click the colour icon and select a bright blue. Click OK and continue adding styles to the input and button. It's easy to experiment with gradients, shadows and border properties.

04 Select both the input and the button (hold Shift while using the Select tool). Press Cmd+G to group the elements together in a container. Containers are much like Photoshop groups, but they have actual dimensions and can receive styling. They also gain abilities like

EXPERT TIP

NUDGE AND PUDGE

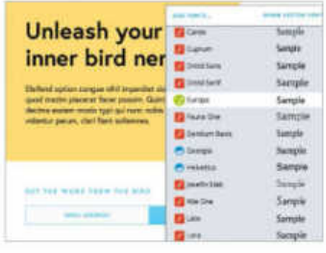
Macaw has two extremely useful commands that make working with a grid a breeze. You can nudge elements around using the arrow keys. However, you can also pudge an element's dimensions by holding Alt while pressing an arrow key. Even more useful, if you hold Cmd+arrow, this will nudge an element left or right by a grid unit, and Cmd+Alt+arrow will resize an element by a grid unit.

auto height, which allows it to shrink or grow based on the rendered height of its children. Change the container's height to auto by clicking the `px` suffix in the `Height` field in the `Dimensions` palette and select `Auto`.

05 Let's add some beautiful fonts using Typekit. First, you will need to create a kit on the Typekit website (*typekit.com*). Be sure to add 'localhost' to the kit's allowed domains.

Back in Macaw, dive into the container you created in step 4 by double-clicking it using the Select tool. This lets you work inside the container without affecting elements outside of it. Select the input and button. Click the `Font Name` field in the Typography palette to open the Font Picker. Click the `Add Fonts...` button and add your Typekit ID in the dialog. Now the Font Picker will include the fonts from your Typekit account.

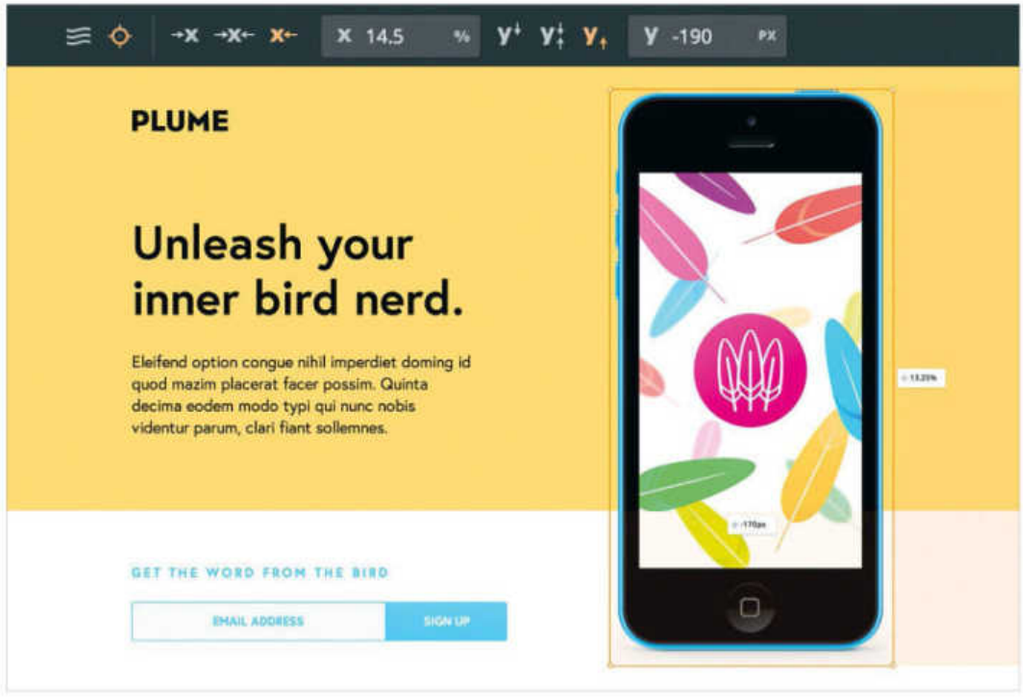
Change the fonts on your form and the text elements in the header. You may need to dive in and out of containers to access these



Type time Add fonts from a Typekit account

elements. Double-click a container to dive in and double-click outside of it to dive out. You can also use the breadcrumb bar at the bottom to jump to specific locations.

06 Let's add some images to that boring header. Click the Library tab. This is where you manage a project's assets. You'll see that there are a few images already in the library. Macaw stores assets in the mcw file so you can pass it along to others without losing references.



Adding images Macaw will automatically convert retina-ready images to make them suitable for high-resolution displays

We'll add the phone to the page. Double-click on the header, then click and drag phone.png from the library to the canvas. This is a retina-ready image and, if you're using a high-resolution display, Macaw will automatically convert it. You can check by navigating to the Image palette in the Inspector. This palette lets you examine the current size of an image in relation to its original size. Make sure that the `@2x` icon is selected.

We want to position the phone so it's always extending below the bottom of the orange section. To do so, we'll use absolute positioning. Select the phone and click the crosshair in the Property bar. Set the x origin to `right` and the y origin to `bottom` using the x and y icons in the Property bar. Now move the phone so that it's positioned to the right of the text and extending below the orange. Even though the header has an auto height, the phone will now extend down as we'd like it to.

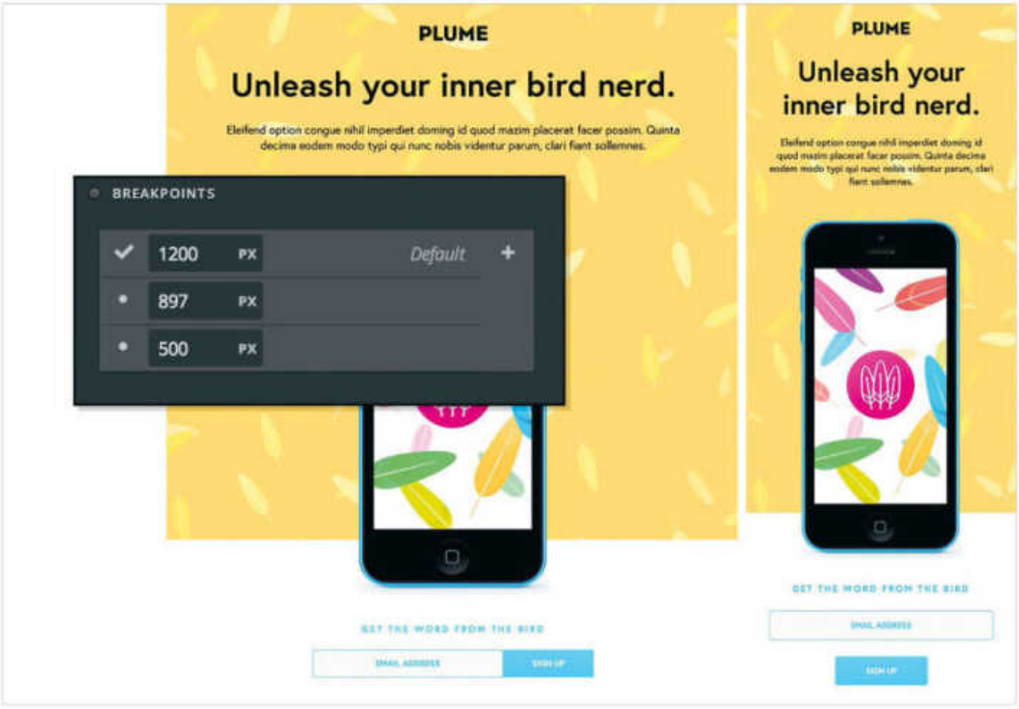
07 We'll now add some pizzazz to the header by adding a background image. Click `body` on the breadcrumb bar. Select the header and click the `+` button in the Backgrounds palette. Choose `Image...` from the drop-down. This opens the background image dialog. Click `Choose from library ...` and select `feather.png`. Set it to cover, using the size drop-down to ensure it always fills the orange header. There are plenty of other properties in here, so feel free to experiment.

08 To make sure the design holds up in different viewports, scroll to the right of the canvas and move the drag handle to the left to resize the canvas and watch your elements flow down the page. Macaw has automatically calculated the document flow, so you can see exactly how your design will behave in the browser.

EXPERT TIP

TEXT SHORTCUTS

Macaw has filler text built in. When editing a text element, type the shortcut 'loremXY' and press `Tab` to quickly insert filler text. Replace the X with a number and the Y with a unit (w for word, s for sentence and p for paragraph). For example, 'lorem2p' will insert two paragraphs. Typesetting is also simple, as calculations such as characters-per-line are visible in the bottom-right of the Typography palette.



Breaking point Breakpoints can be added wherever you like – alter your layout, and toggle between breakpoints on the Breakpoint palette

EXPERT TIP

INTERACTIVITY

For interactions, add a variable name to an element in the Inspector palette and Macaw will populate a scripting variable for it. In the Scripts palette you'll see a list of variables. Write in any JavaScript or jQuery, and Macaw will wrap your scripts in an anonymous function, populate the necessary scripting classes and generate selectors to make your scripts work.

Let's add a breakpoint. While resizing the canvas, you'll see a tooltip appear, indicating the width. Drag the handle down to a width of 900px and press `Cmd`. Click `Yes` in the alert to insert a breakpoint. You can now modify your design to cater to this specific width. Layout can be changed and styles can be reworked. You can easily toggle between breakpoints using the Breakpoints palette or by hovering breakpoints on the ruler.

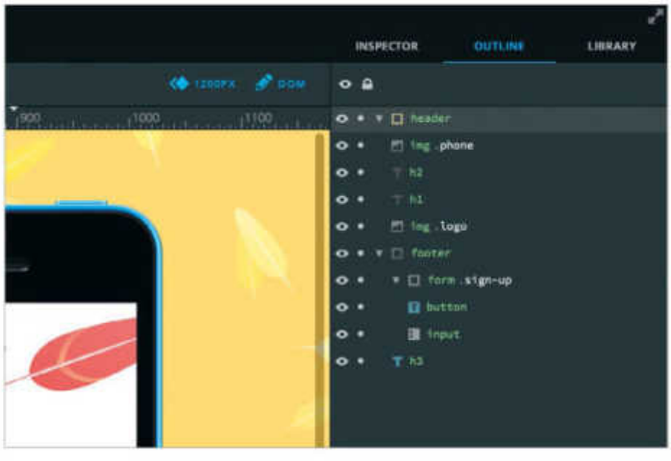
When working with breakpoints it's important to understand that properties trickle. When you change a property, it's applied to the current breakpoint. That value will trickle to the breakpoints below it, but it does not travel upwards. If you have different values set for a property across breakpoints, Macaw will outline the field in blue. When you hover over one of these fields, a property table will display showing the values on each breakpoint. You can quickly grab a value and apply it by clicking on one of the values in the table. You can also distribute a value to all breakpoints by `Cmd+clicking` it. Go ahead and add breakpoints wherever you think the design starts to break down. I added them at 900px and 500px.

09 Macaw uses a powerful design-to-code engine to convert your document into clean, succinct HTML and CSS. Macaw will handle the heavy lifting for you, but it needs some help first.

Click the `Outline` tab. Here you can rename and organise elements in a manner similar to Photoshop's Layers palette. Renaming elements plays a key part in publishing, because it defines the semantics and class names to use.

Macaw uses a simple dot-syntax naming scheme. The first word used in the name is parsed as a tag name. If that tag exists in the HTML spec, it will be highlighted in green and used when publishing. Macaw then looks for a full stop followed by a class name. This is a simple and way to define the semantics inside your document. Select the container element you created earlier, double-click its name in the outline and rename it 'form.sign-up'. You'll see that `form` is highlighted in green because it's a valid HTML tag.

10 Our design is optimised for all breakpoints and elements are all named semantically. You can publish your document at any time by pressing `Cmd-P`. This will generate all the HTML and CSS for your project and open the preview browser. From the preview window you can ensure proper rendering, view other pages and inspect the generated code. Macaw also has a built-in feature called Remote Preview, which enables you to broadcast your design to other devices on the same wireless network. Simply navigate to the IP address shown in blue in the preview window's address bar and Macaw will automatically reload updates made to the published project to that device.



Transformation A powerful design-to-code engine will turn your designs into HTML and CSS

ABOUT THE AUTHOR

KAMIL ZIEBA

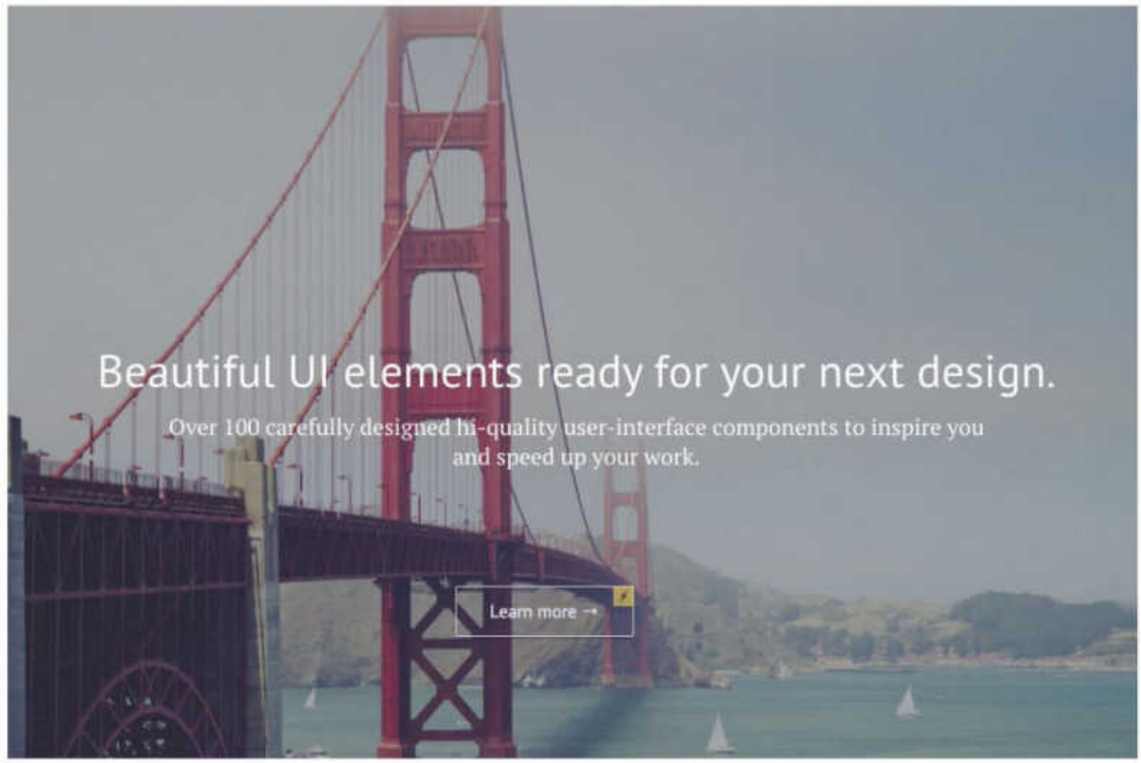
w: *uxpin.com*

t: @ziebak

job: Chief product officer, UXPin

areas of expertise: UX, visual design, user research, frontend development

q: what's your favourite sweet treat?
a: That would be my girlfriend's homemade brownies



UXPIN

TRANSFORM PSDS INTO LAYERED PROTOTYPES

Kamil Zieba explains how UXPin can help you create rich mockups from Photoshop files, without using code or rebuilding your designs

Wireframes and mockups show you how a design looks, but prototypes demonstrate how a design works. Prototypes let you experience the design the same way a user does. Simply put, prototyping is the ultimate reality check.

Modern designers may follow different processes, but at one point they all need to build some sort of mockup. But the prototyping process is far from perfect – most available tools mean that you'll need to compromise in some way.

If you want a quick prototype, you'll have to sacrifice interactivity, as most solutions flatten your files. Animations and interactions are the lifeblood of modern design, so this is far from ideal. If you want a rich prototype, you'll need to invest time in working on code. Again, not ideal.

However, there is one tool that enables designers to build prototypes without compromising on either of these fronts: UXPin (*uxpin.com*). We built UXPin to adapt to a range of design approaches, whether that's going from low-fi to hi-fi in UXPin or using it to add interactivity to Photoshop or Sketch files.

In this tutorial, I'll show you how to move an existing PSD for a website design into UXPin, and add some simple interactions to the layer groups. When we're done, you'll have a high-fidelity prototype that looks and acts just like a real website. You can see what we're aiming for at *netm.ag/prototype-267*.

VIDEO

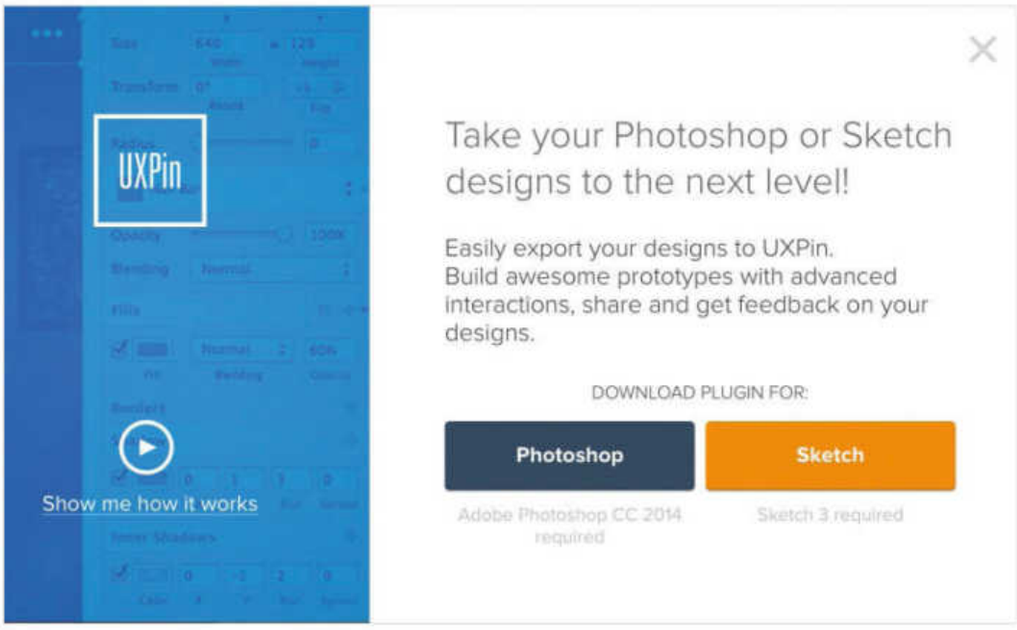
The UXPin team has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/uxpinvid-267*

01 For this example we will import an existing Photoshop file from our free web UI kit (*netm.ag/kit-267*) into UXPin. The first step is to log in to UXPin and install the Photoshop plugin in the bottom left-hand corner.

02 Open Photoshop, select the elements you want to export and click `File > Generate > UXPin Export`. We're going to export all the elements to UXPin, to ensure we will be able to add interactions to any existing layers. The file will likely export to the desktop, and will have a '.uxpin' extension.

03 Return to UXPin and find your project. Drag and drop your '.uxpin' file into your project view. Now your exported file has been successfully uploaded into UXPin, and you're ready to turn it into a prototype.

04 When you open the file in UXPin, you'll see that all the Photoshop layers have been preserved. Feel free to click around and add some interactions. This is pretty easy to do – just click on any object on screen, then click on the 'lightning bolt' icon to choose your interactions.



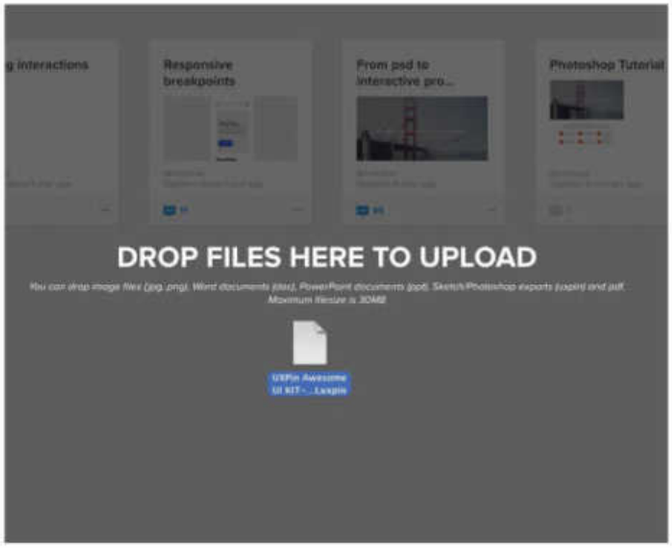
Step 1 Log in to UXPin and download the Photoshop plugin via the button on the homepage

05 To set your first interaction, click the 'Interactions' button. The 'Link' button is a shortcut that enables you to link UI elements to pages. Below that, 'Recently created interactions' is a list from which you can quickly jump to any recently created interaction in your design.

06 Setting up interactions for a given UI element is a three-phase process: first choose what should trigger the action, then what type of action the trigger should initiate, and what element should be affected by it. There are two ways of picking the element that should be affected by the interactions – you can either choose it from the list of all elements, or use the crosshair to point it out on the canvas.

07 In UXPin, there is a vast number of triggers, interactions and animations you can choose from. Triggers include Click, Double-click, Hover, Mouse in, Focus, Key press, Window is scrolled to, and Page is loaded. Actions include Show element, Hide element, Toggle visibility, Go to page, Scroll element, State: enable, Move by, Rotate element, Change opacity and Change style. Finally, animations include Linear, Ease in, Ease out, Ease in-out, Fade and Slide.

08 For every interaction you can choose a time delay before it occurs. For more precision, some interactions have ▶



Step 3 Upload your exported PSD designs into UXPin

EXPERT TIP

COLLABORATING ON DESIGNS

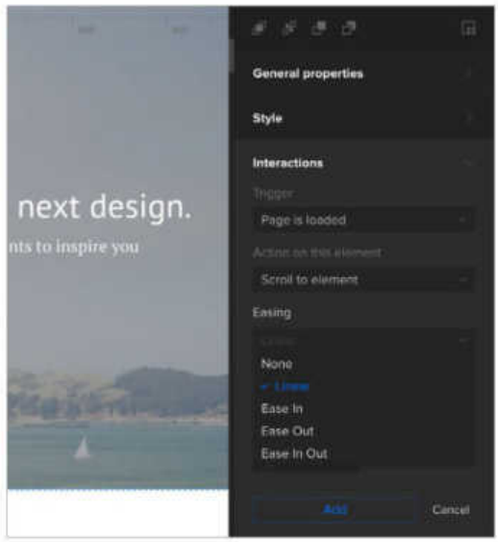
Need to get feedback from the team? As you iterate your designs, make sure you click the 'Simulate design' button in the top right corner to generate a preview link. Right after you click the button, type in the emails of the people you wish to notify. Your team will now be able to comment on your designs, and a daily summary of progress will be automatically emailed each day.

additional settings, like the amount of pixels you want an element to move by, or how much time an animation should last for.

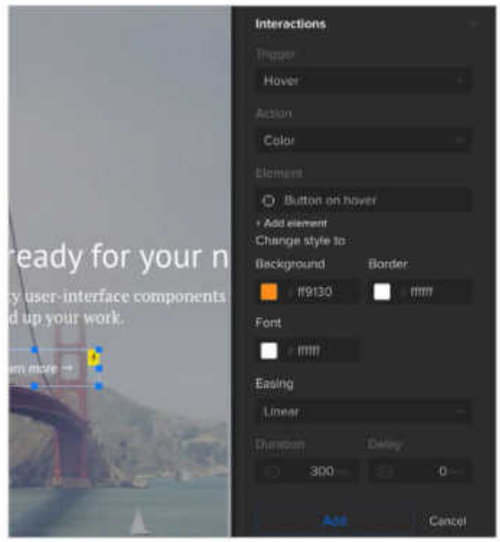
09 In this tutorial, we'll add a few simple interactions and animations to our uploaded PSD file. On hover, the 'Learn more' button will change from translucent to orange. On clicking the button, the user will be scrolled down to the sign-up form at the bottom of the page. The sign-up form will also appear when a user scrolls down the page. Finally, after completing the form, a 'Thank you' message will appear.

10 First, let's add a interaction to the 'Learn more' button. We'll set up our interactions so when we hover over the button, it eases into a different colour. Set your trigger to `Hover` and set the button action to `Change style`. Now, let's select the new style for the button – we'll use orange. To make the colour appear smoothly, set the animation to `Ease in`, with a delay of 300ms.

11 Now we need to add another interaction to make the page scroll to the next section when the user clicks the orange 'Learn More' button. Set your trigger to `Click`, then set the button action to `Scroll to element`. Now, just click on the sign-up box to designate it as the 'scrolled to' element. The drop-down menu will automatically update with the name of the element ('Box #7858').



Step 7 Add interactions to your elements by selecting from the extensive range of triggers, actions and animations



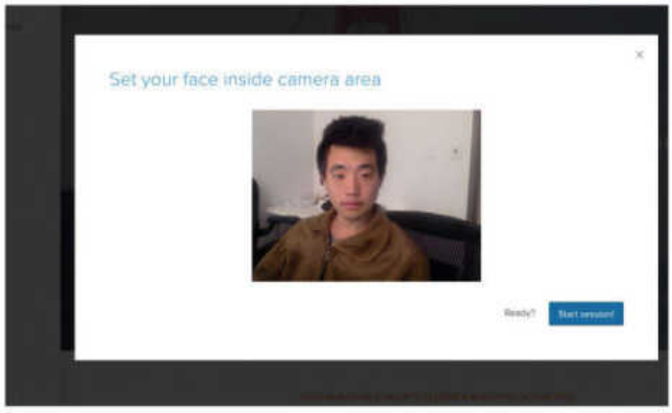
Step 10 For your 'Learn more' button to turn orange on hover, your interaction panel should look like this

12 To make the scroll feel nice and smooth, let's set the animation to `Ease in`, with a delay of 300ms. Click the first button in our prototype (*netm.ag/prototype-267*) to see the effect we've created.

13 Next we will take care of the sign-up form. I'll show you how make it smoothly appear when the window is scrolled to a certain level. Because we want the form to appear following a certain action, you'll need to make sure it's hidden by default. Now, select the whole layer of the form and click the 'lightning bolt' icon for interactions. The trigger will be `Window is scrolled to` and the action will be `Show element`.

14 As we've introduced a form in our prototype, why not make the inputs interactive to complete the experience? Choose inputs from the library of elements (`cmd+F` and type 'input'), style it accordingly and place it where you want customers to sign up. You don't need to set any interactions to enable users to type inside the inputs or switch to the next one using the tab key – those features are automatically included.

15 As the final touch we'll introduce a sign-up confirmation. We'll be setting interactions between three elements: the 'Sign up' button, the sign-up form and the 'Thank you' slide. However, we only need

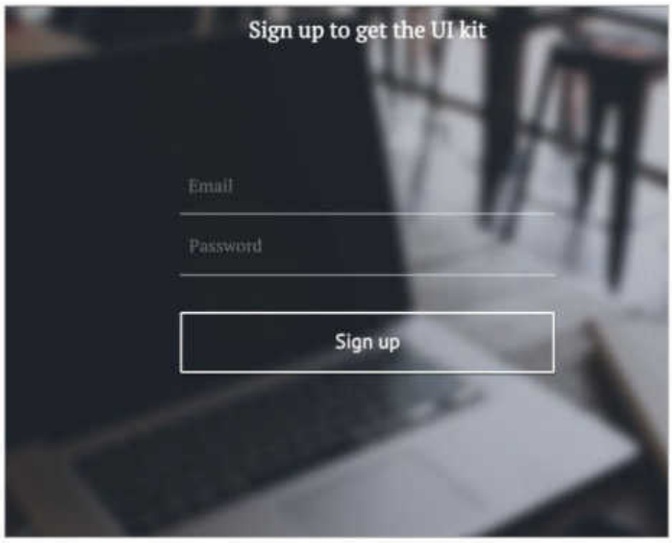


Usability testing Watch others interact with your design using UXPin's testing features

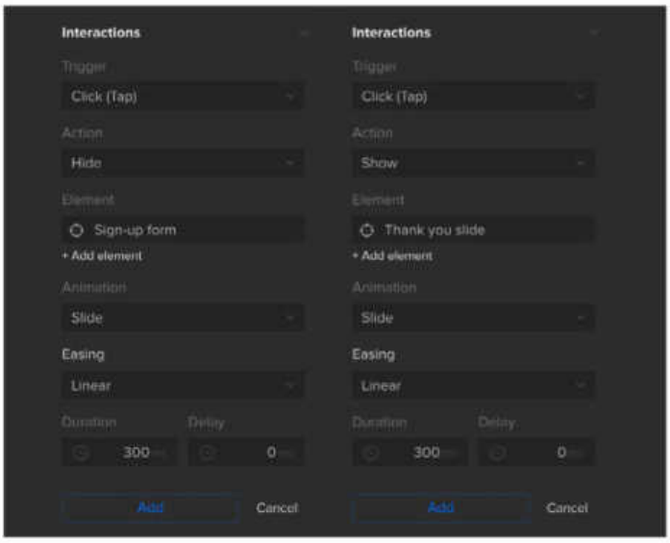
EXPERT TIP

USABILITY TESTING

Usability testing is the best way to support design decisions. UXPin's built-in usability testing lets you create tasks, record user reactions and generate video clips for presentations. In the Preview mode, just tick the webcam icon in the top right-hand corner to create tasks. Next, type in the email addresses of your users and they will be notified. Once you start the session, you can speak to them through the app, which records all clicks, facial reactions and audio.



Step 15 Set your interactions so that when a user successfully signs up, the form slides out, and is replaced with a friendly 'Thank you' message



Step 16 Fill out your menu to add interactions to hide the 'Sign-up form' (left) and show the 'Thank you' slide (right)

to set up the interaction on the 'Sign up' button, because this will trigger the actions for the other two elements. When clicked, the 'Sign up' button will hide the sign-up form and the 'Thank you' slide will appear in its place.

16 To hide the sign-up form when the 'Sign up' button is clicked, make sure your interaction menu is set to `Trigger: Click`, `Action: Hide element` and `Element: Sign-up form`. To show the 'Thank you' slide when the 'Sign up' button is clicked, make sure your interaction menu is set to `Click`, `show element` and `Thank you slide`.

17 Now you're all done! Your 'Learn more' button will scroll the user down to the input form, which transitions to a 'Thank you' page upon completion. If you want to create responsive versions of the design, just click on the `Add responsive version` tab in the bottom-left corner. You can set breakpoints for mobile, tablet and custom resolutions, and it's simple to copy over all your elements to the new, responsive version.

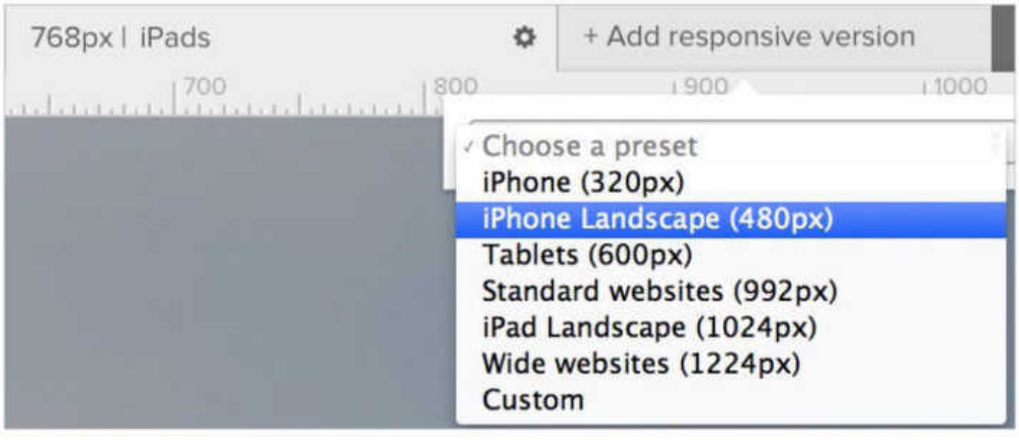
18 For example, if you select 'iPads (768px)', you can immediately see the screen limits in the new version of the design. The existing layout will obviously need to be modified for the new screen dimensions. Tweak these as needed, and you're all set.

19 So there you have it, a comprehensive guide to turning a flat PSD into an interactive prototype, with responsive breakpoints. You can explore all these features and more by signing up for a free 30-day trial on UXPin.

EXPERT TIP

LIVE PRESENTATION

Ready to present your new designs? UXPin's Live Presentation tool links your stakeholders to your project for live presentations. Not only is there voice calling and cursor tracking, but viewers can follow along as you focus on elements or switch pages. If you need to provide broader context, you can also toggle the sitemap to show them the flow of the experience.



Step 17 You can add responsive breakpoints for a range of different devices

View source files here!

All the files you need for this tutorial can be found at *netm.ag/foundationgit-266*

ABOUT THE AUTHOR

BRANDON ARNOLD

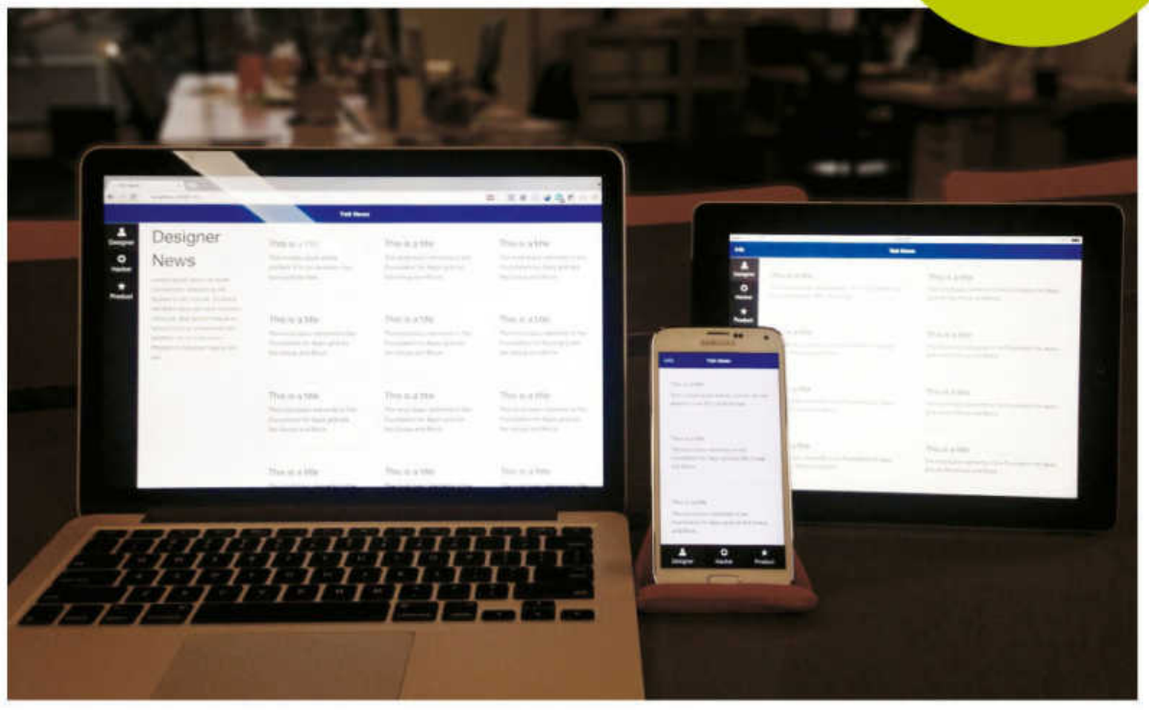
w: *brandon-arnold.com*

t: @brandon_arnold

job: Design lead, ZURB

areas of expertise: Product design, responsive design, HTML, SCSS

q: what's the worst holiday you've ever been on?
a: I booked a wine tasting through a discount site. We ended up in a musty workshop, sipping wine and eating individually wrapped cheese slices



FOUNDATION

DESIGN A PROTOTYPE FOR A RESPONSIVE WEB APP

Brandon Arnold demonstrates how to mock up a fully responsive web app layout using ZURB's Foundation for Apps framework

Over the past several years, the web has changed. We're racing away from an advertising web that discusses things, to a web of doing and creating things. We're transitioning to a more app-centric web. People want to build fullscreen, immersive experiences in the browser, and they want these apps to work on any device.

This is why we at ZURB created Foundation for Apps (*foundation.zurb.com/apps*), the first frontend framework created for developing fully responsive web apps. Over the course of this tutorial I'll show how to use our new grid to create an app layout that responds for every device. We'll prototype Angular routes and use our motion classes to add slick animations to our views. From there it's easy to take this prototype and load in live data.

PLANNING THE APP

We'll be creating a prototype of a simple news aggregator app that could eventually pull in articles from things like Designer News, Hacker News and Product Hunt. Each source will live in its own view, but will share a common header.

We'll start with a simple sketch for a mobile and desktop view (top right). We can see that on mobile our app will follow a common theme, with tabs at the bottom that enable the user to swap between each news source, a list of the most popular articles, and some information about that news source in a side panel. In contrast, on the desktop version this aggregator will feature the navigation along the left side, the popular articles along the right, and the content on the panel in-between the two.

VIDEO

Brandon Arnold has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/foundationvid-266*

It would be difficult to accomplish this distinction between desktop and mobile using a traditional grid system. However, Foundation for Apps was created to solve these types of problems easily.

CREATING A NEW PROJECT

The Foundation CLI makes it easy to install new Foundation for Apps (F4A) projects, and will make it even simpler to update when patches and upgrades arrive. You'll install the CLI once, then you can create hundreds of new projects with one simple command. You'll need to ensure you have Ruby, Node.js and Git installed on your machine. For information on installing the CLI, visit *netm.ag/install-266*.

Once the CLI is installed, creating a new app is as easy as entering a single command. Let's call our app 'yetinews'.

```
foundation-apps new yetinews
```

You'll be greeted by ZURB's yeti, who will spin up a new project for you. This can take anywhere from 30 seconds to a couple of minutes, so sit tight and wait

We'll prototype Angular routes and use motion classes to add slick animations to our views

for the yeti to tell you he's all finished. Once your app has been created we'll `cd` into that directory:

```
cd yetiNews
```

Then we'll start our app:

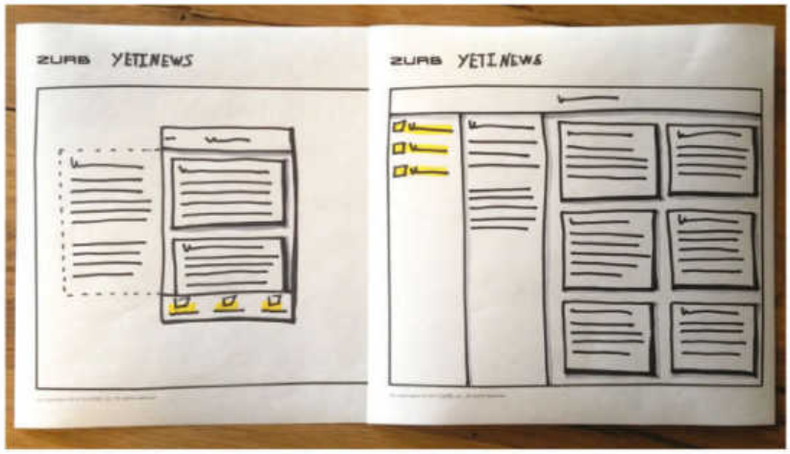
```
foundation-apps watch
```

This runs our gulp task, creates some routes and starts up a server on your machine. You can view our new app by visiting `localhost:8080/`.

CODING MOBILE-FIRST

When we open our newly created project, we can see several files and folders in the base directory. Let's head straight over the 'client' folder and take a look at what we've got there.

If you're already a Foundation user you'll see some familiar things – like 'app.scss' and 'app.js', where you put all your custom Sass and JavaScript. There's the '_settings.scss' where you can change nearly every style in the framework, and some



Early concepts A sketch showing what our app will look like on smaller mobile screens and larger desktops

template pages and the 'index.html'. Let's start by opening 'index.html'.

F4A supplies a sample page for your new project, but since we're learning how to build an app, we'll start by nuking everything between the `body` tags. After the `body` tag we'll create the `grid-frame`. This contains the entire app and, by setting a defined width (100 per cent) and a defined height (100VH), it allows us to easily carve the screen up vertically.

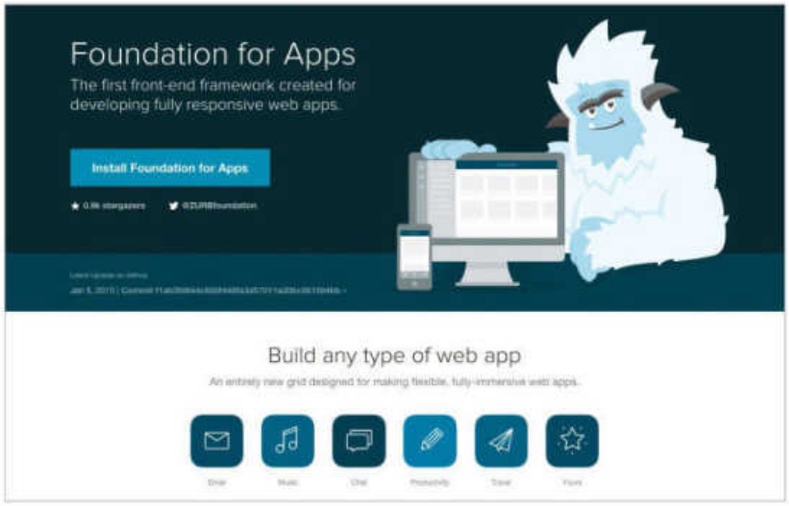
```
<body>
  <div class="grid-frame">
  </div>
</body>
```

Now we'll create the header and main content area of our app. We'll introduce `grid-blocks`, `grid-content`, the `vertical` grid and the `shrink` class.

```
<div class="grid-frame vertical">
  <div class="grid-content shrink">
  </div>
  <div class="grid-block">
  </div>
</div>
```

Let's break down these new elements:

- `vertical`: This can be applied to `grid-frames` and `grid-blocks` to force their direct children to be displayed vertically as opposed to horizontally
- `shrink`: By default, `flex-box` wants to spread elements out evenly across the available space. The `shrink` class tells the element to only be as large as it needs to be. Items without a `shrink` class will take up the remainder of the space
- `grid-block`: These are roughly analogous to rows, but unlike rows they can be sized. Mainly, they



Introducing F4A ZURB's new framework enables users to build web apps that work on any device

- are items set to flex, allowing every item inside of them to be aligned without floats
- `grid-content` : This is used when you want to put regular content (`<p>` , `<h1>` and so on) in your app. They can be sized or not sized. In the case of the latter, the grid will attempt to fit the content inside of them evenly across their parent

We need to name our app, as well as including some actions at the top of the page. We'll add a `title-bar` to the app to quickly create a simple header:

```
<div class="grid-frame vertical">
  <div class="grid-content shrink">
    <div class="primary title-bar">
      <div class="center title">Yeti News</div>
      <span zf-toggle="info" class="left hide-for-large"><a
href="#">info</a></span>
    </div>
  </div>
  <div class="grid-block">
  <!-- main-content -->
  </div>
</div>
```

This creates a basic header, with a link titled 'info' that we'll use to toggle an info panel. Foundation uses `zf-toggle` to call panels, modals or off-canvas menus based on their `id` . When the user clicks this on smaller screens, we want our info panel to pop out. On larger screens, we'll need to hide this panel, as its contents will already be displayed.

We now have the basic header for our app, so let's build out the content area and navigation. We'll start by creating our info panel, then a series of cards that will act as our news articles, and finally the navigation.

We'll start inside the block we created underneath our header:

```
<div class="grid-block vertical">
  <!-- main content -->
  <div class="grid-block">
    <div zf-panel position="left" id="info">
      <div class="grid-content">
          <h1>Designer News</h1>
          <p>Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nullam in dui mauris. Vivamus hendrerit arcu
sed erat molestie vehicula.</p>
      </div>
    </div>
    <div class="small-12 large-9 grid-content">
      <!-- our feed -->
    </div>
  </div>
  <div class="grid-block shrink">
    <!-- our navigation-->
  </div>
</div>
```

As our block is vertical, we can stack our content and navigation on top of each other. Here, we have added a panel into our app using `zf-panel` – the F4A Panel Directive – and given it the position it will animate from.

Next let's add our content blocks in cards:

```
<div class="grid-block vertical">
  <!-- main content -->
  <div class="grid-block">
    <div zf-panel position="left" id="info">
      ...
    </div>
    <div class="small-12 grid-content feeds">
      <div class="grid-block small-up-1">
        <div class="grid-content">
          <div class="card">
            <div class="card-section">
              <h4>This is a title</h4>
              <p>This is really great article content. It's not
dynamic now, but could be later.</p>
            </div>
          </div>
        </div>
        <!-- more cards -->
      </div>
    </div>
  </div>
  <div class="grid-block shrink">
    <!-- our navigation-->
  </div>
</div>
```

To size our feed we've used the parent sizing grid, and to style individual items in the grid we've used our card components. Parent sizing allows us to declare the size of direct child elements on the parent when we know how many items we want in each row.

Here we add `small-up-1`, because on small screens we only want to display one item in each row, but on larger screens we'll want more. We then use the card component to create clean-looking box dividers that hold our article title and description.

Next we'll add our navigation:

```
<div class="grid-block vertical">
  <!-- main content -->
  <div class="grid-block">
    ...
  </div>
  <div class="grid-block shrink">
    <ul class="menu-bar icon-top dark">
```

F4A's variables enable users to change just about anything about the look of their app

```
        <li><a><img zf-iconic="" data-src="assets/img/
iconic/person.svg"> Designer</a></li>
        <li><a><img zf-iconic="" data-src="assets/img/
iconic/cog.svg"> Hacker</a></li>
        <li><a><img zf-iconic="" data-src="assets/img/
iconic/star.svg"> Product</a></li>
      </ul>
  </div>
</div>
```

We used the `menu-bar` component to make a tab list for our app, and added some iconic icons to make it feel more like an app tab bar. The `menu-bar` has tons of customised options. By default it's horizontal, but it can also become vertical.

This gives us the main layout for our app on a mobile screen. Next we'll take a look at how to adapt this layout on a desktop screen.

ADAPTING FOR DESKTOP

We've created the layout for our mobile site, but we want it to display differently on a desktop. We'll make the following changes to our code to get it to properly match the sketch we created earlier. Here are some areas we'll change to make our layout match:

FOCUS ON

STYLING FAST WITH SASS

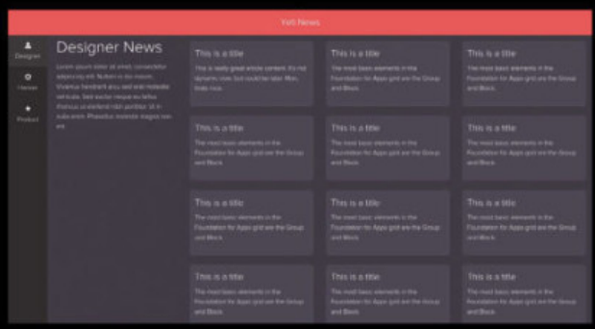
Foundation for Apps, like its 'For Sites' counterpart, comes with tons of variables to enable users to change just about anything about the visual look of their apps.

To quickly change the look we'll head over to the settings file, which can be found at `client > assets > sass > _settings.scss`. Once open, we can begin changing variables.

Some of the most high impact changes include:

- `$rem-base`: This controls the base font size, and can shrink or enlarge the text and spacing within your app
- `$body-background`: This is used to change the background colour of your app
- `$body-font-color`: This can change the default text colour for `p` tags and `h1` to `h6` tags as well. Swapping two variables here could easily convert your app to a dark theme
- `$primary-color`: This is the main colour of your app and is used for all links and buttons. Many components have a primary class that, when applied, can turn them from a plain white to a theme based on the primary colour

There are plenty of variables for every component, so you'll rarely need to override the existing styles. Not only does this make creating your app easier and quicker, it also prevents duplicate code from being written just to style your app.



Fast work In this tutorial, we changed a few variables and easily created a distinct style for our app, all in about 15 minutes

RESOURCES

ADDITIONAL RESOURCES

Foundation for Apps offers places where users can get help or access additional resources. Below are some of the URLs to check out for more information on the Foundation family and ZURB as a whole.

Foundation for Apps Docs (*foundation.zurb.com/apps/docs*) – For more information regarding F4A components and directives, visit the Foundation for Apps Docs

Foundation Docs (*foundation.zurb.com/docs*) – To build marketing sites and landing pages, view information for Foundation for Sites by visiting the Foundation Docs page

Ink (*zurb.com/ink*) – To build HTML emails using a ZURB framework, check out Ink, our Foundation for Emails framework

Foundation Forum (*foundation.zurb.com/forum*) – If you get stuck or need help with any of the ZURB Frameworks make your way to the Foundation Forum

Foundation Training (*foundation.zurb.com/learn/training.html*) – If you'd like to learn Foundation from the people who create it, take a look at our Training pages

GitHub (*github.com/zurb/foundation-apps, github.com/zurb/ink, github.com/zurb/foundation*) – Any other questions can be directed to the respective ZURB GitHub accounts

Twitter – For quick queries, tweet @ZURBFoundation

Yetinews When installing a new Foundation for Apps project, you'll be greeted by our friendly yeti to alert you of progress

```
<div class="grid-block vertical medium-horizontal">
  <!-- main content -->
  <div class="grid-block medium-order-2">
    <div zf-panel position="left" id="info" class="large-3
large-grid-content">
      <div class="grid-content">
        ...
      </div>
    </div>
    <div class="small-12 large-9 grid-content">
      <div class="grid-block small-up-1 medium-up-2 large-
up-3">
        ...
      </div>
    </div>
  </div>
  <div class="grid-block shrink medium-order-1">
    <ul class="menu-bar icon-top dark">
      ...
      </ul>
  </div>
</div>
```

We have introduced a few new classes to help make our layout adapt to large screens – things such as `medium-horizontal`, `large-grid-content` and `medium-order-1`.

Let's take a look at these:

- `[size]-horizontal`: The same way we can tell a grid to be vertical, we can also tell it to go back to being horizontal at a certain breakpoint. In this case we're telling the grid to be vertical, but on medium screens and larger, we want it to switch and become horizontal
- `[size]-grid-content`: There's a panel that is off-screen on small devices, but on larger screens we want that panel to be in line with the content. By adding a class to the panel to tell it when to be `large-grid-content`, we're able to make it part of the grid on large screens
- `[size]-order-x`: We knew we wanted the navigation to be on the bottom on mobile devices, but on the left of the screen on medium devices and above. To achieve this we'll use F4A's source ordering classes, which let us declare which items will display first on screen, regardless of what order they sit in the code. We used `medium-order-1` on our navigation to tell it to come first on screens above a medium size, and `medium-order-2` to tell our content to come second

Now we have a unique layout that works on all size devices, takes advantage of the extra space on larger screen and uses the same markup for all.

ANGULAR ROUTING AND VIEWS

So far all our content lives on 'index.html'. However, we're creating a single-page web app, so we want to make the navigation bring up different content in the main content area.

A basic website is a collection of HTML files connected by links. Single-page apps work slightly differently – all your site's logic is handled on one file: 'index.html'. This contains the core elements of your app, such as the header, navigation and footer. The individual pages are stored as separate HTML files in a templates folder.

Now, let's put together a view for our app. Earlier, we created a `grid-block` for our main content area. Let's use that block as the container for our view.

```
<!-- main content -->
<div class="grid-block medium-order-2" ui-view>
...
</div>
```

We have added a `ui-view` to our `div`. This will tell Angular which area of the page should be swapped

We have introduced a few new classes to help make our layout adapt to large screens

out. We'll need to take all the content inside that `div` and move it to a separate file, so cut that code and we'll use it in a second.

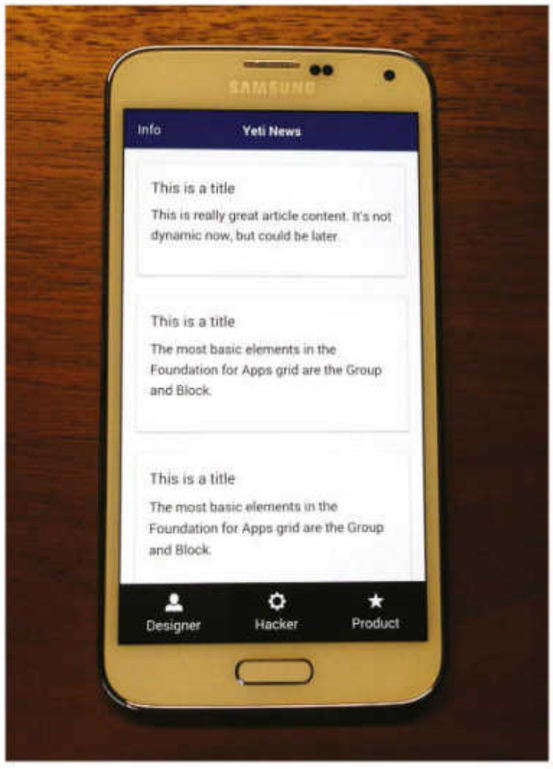
In our 'templates' folder we already have a file named 'home'. We'll use that file as the basis of this view. In F4A we've created a system for creating routes that doesn't require the user to know Angular or UI-Router. We use `Front-matter` at the top of our template files and then our gulp task parses that out as a route.

In our 'home.html' file we'll keep everything within the dashes, but replace the HTML under it with the HTML we just cut from the previous page.

Next we'll look at the `Front-matter`.

```
---
name: designer-news
url: /
animationIn: hingeInFromLeft
---
```

The `Name` is how we'll reference our app, and the URL is what the '/' in our URL will turn into. While



Mobile view The completed mobile app, ready to be built out for larger screens

those are the only two required parameters to make the views work, we'll also add an `animationIn` option to animate the view as it comes in.

For this app we'll call this view 'designer-news', with just a slash for the URL (because we want this to be the default view) and a `hingeInFromLeft` class from F4A's motion UI. We can create more pages in the 'templates' folder, give them their own names and their own URLs.

Lastly, let's link our views to our navigation. We'll use `ui-sref` as opposed to `hrefs` and call the links by the names we gave them in each template.

```
<ul class="menu-bar icon-top dark">
  <li><a ui-sref="designer-news"><img zf-iconic="" data-src="assets/img/iconic/person.svg"> Designer</a></li>
  <li><a ui-sref="hacker-news"><img zf-iconic="" data-src="assets/img/iconic/cog.svg"> Hacker</a></li>
  <li><a ui-sref="product-hunt"><img zf-iconic="" data-src="assets/img/iconic/star.svg"> Product</a></li>
</ul>
```

There we have it – a complete prototype for our news app. We have looked at swapping from a vertical grid to a horizontal one, and featuring a slide-in panel content that becomes inline on larger screens. While this app is pretty basic, it has many of the same principles of a more complex one, and would be ready for you or a backend dev to hook up dynamic data.

RESOURCE

ZURB offers free product design lessons, ranging from sketching interfaces to the right way to ask for feedback: *zurb.com/university/lessons*

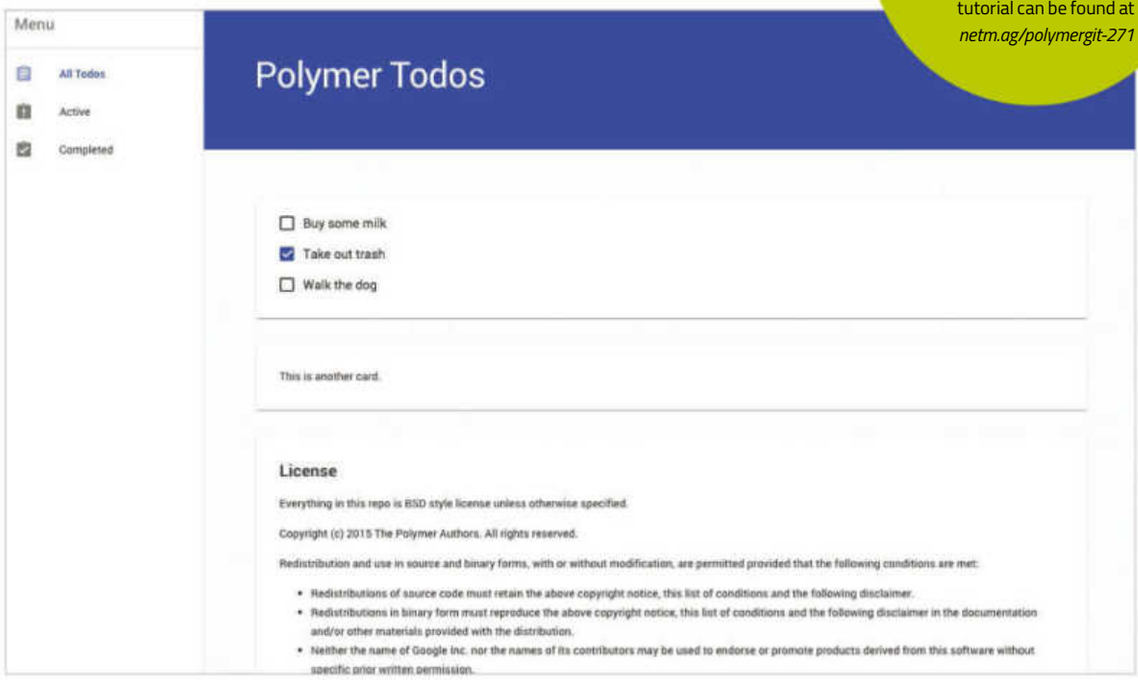
View source files here!

All the files you need for this tutorial can be found at *netm.ag/polymergit-271*

ABOUT THE AUTHOR

ROB DODSON

w: *robdodson.me*
t: @rob_dodson
Job: Developer advocate, Google
Areas of expertise: Web development, HTML, CSS and JS, web components
q: what's the longest you've gone without a computer?
a: That's crazy talk



POLYMER

USE POLYMER TO BUILD A MATERIAL DESIGN APP

Google's **Rob Dodson** explains how to start building your first material design app with Polymer 1.0 and the new Polymer Starter Kit

At last year's Google I/O, the Polymer team announced the release of Polymer 1.0 (*polymer-project.org/1.0*). At a third of the size of previous versions, and running up to five times faster, the 1.0 release is a signal that it's time for us to start using Polymer in production. And getting started with the library has never been easier, thanks to the new Polymer Starter Kit (PSK), an opinionated scaffold for building Polymer apps.

In this tutorial I'll walk you through how to create a simple to-do list using PSK, highlighting some of the amazing goodies that come with it.

GETTING SET UP

As a prerequisite, make sure you have a current version of Node (*nodejs.org*) installed, as you'll be using npm in a few spots. Next, grab the starter kit from its official site at *netm.ag/PSK-271*, or if you're into Yeoman, you can install the Polymer generator (*github.com/yeoman/generator-polymer*) which uses the starter kit.

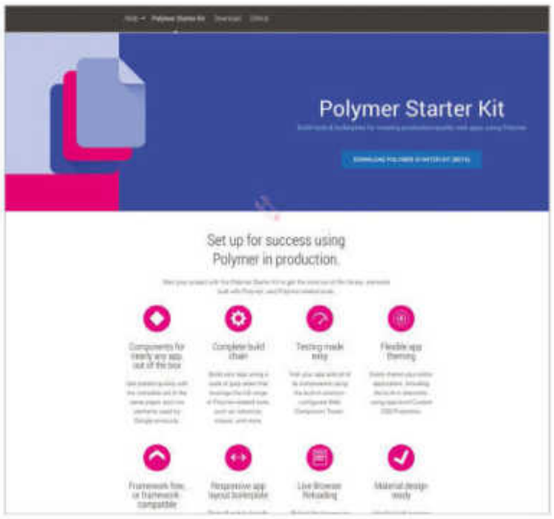
```
npm install yo generator-polymer -g
```

If you're downloading PSK from the official site, I recommend going for the 'Intermediate - Advanced users' version, as it comes with its own gulp server and Browsersync for rapid testing. The Yeoman generator outputs this version by default.

Once you've downloaded the zip, unarchive it and `cd` into the directory from your terminal. You'll need to install your package dependencies, so from the root of the starter kit execute the following command in your terminal:

VIDEO

Rob Dodson has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/polymervid-271*



PSK If you're new to Polymer, the Starter Kit will get you off on the right foot

```
npm install && bower install
```

Next, fire up the gulp server to preview the app in your browser.

```
gulp serve
```

Out of the box, PSK comes with a layout based on the material design spec for adaptive UI (*netm.ag/adaptive-271*), as well as an already configured client-side router. Try clicking some of the items

The Polymer Starter Kit is an opinionated scaffold for building Polymer apps

in the menu on the left-hand side and you'll notice that the page changes, as does the URL in the address bar.

PSK also ships with a few material design custom elements. For the documentation for these elements, and to browse the entire catalogue of what's available, go to *elements.polymer-project.org*.

CREATE A TO-DO ITEM

Let's start by creating a new Polymer element for our to-do items. Create a new folder in the 'app/elements' directory called 'todo-item' and place a 'todo-item.html' file inside.

Every element's definition will begin with a list of imports at the top of the file. These ensure that any

FOCUS ON

THE CATALOGUE

To make discovering elements easier, Polymer has launched a new catalogue (*netm.ag/elements-271*). Elements are split into product lines based on features and appearance.

Iron elements
Formerly called core elements, these are building blocks for many of the more advanced components. They often contain behaviours that you would want to use to augment an element. For example, `iron-collapse` provides a basic collapsible container that can be composed inside a more advanced component like an accordion or drop-down menu.

Paper elements
These are Polymer's embodiment of the material design specification. All of the UI that you see in the PSK sample app is built with these elements.

Web components
The Google web components are Google services bundled up as easy-to-use tags. We won't cover them in this tutorial but they're very fun and easy to use, and perfect for mashups.

Gold elements
The gold elements are dedicated to ecommerce and improving the experience of inputting customer data and checkout flows.

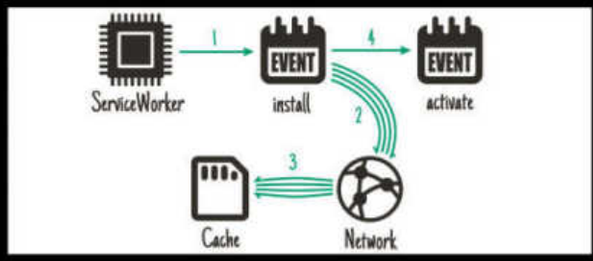
Neon elements
These are for transitions and advanced animations. The element formerly known as `core-animated-pages` now lives in this line.

Platinum elements
Platinum elements provide offline support through Service Worker. Using these elements you can quickly cache your entire application so it works without a network. Very powerful stuff.

Molecules
These represent third-party libraries wrapped as web components.

IN-DEPTH

GET TO KNOW SERVICE WORKERS



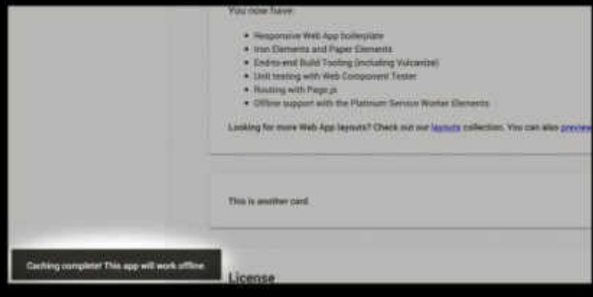
Offline cookbook A visual demonstration of Service Workers from Jake Archibald's offline cookbook

Offline caching, background sync, geofencing and push notifications sound like features only available to native apps, but they are actually landing in browsers today, thanks to a new technology called Service Worker. And you can take advantage of them with the Polymer Starter Kit.

A Service Worker is a script run by your browser in the background, separate from the current web page. In current versions of Chrome you can leverage this script to intercept network requests and serve resources from a cache, or to handle push notifications and provide timely updates to your users.

PSK adds support for these features with a set of elements called platinum elements. These will actually cache your entire application so it runs offline, without you needing to write a single line of code. Pretty amazing.

To learn how to enable the platinum Service Worker elements in your app, look at this section of the starter kit README (*netm.ag/readme-271*). Also, be sure to read through the catalogue section on platinum elements to learn everything that Polymer and Service Workers make available to you (*netm.ag/catalogue-271*). Finally, get to know the underlying Service Worker technology by reading Matt Gaunt's summary on HTML5Rocks (*netm.ag/gaunt-271*) and Jake Archibald's offline cookbook (*netm.ag/archibald-271*).



Going offline Enabling a PSK app for offline is as easy as uncommenting a few lines of HTML and JavaScript

dependencies are loaded before the element attempts to boot up. Add the following imports to the top of your 'todo-item.html' folder:

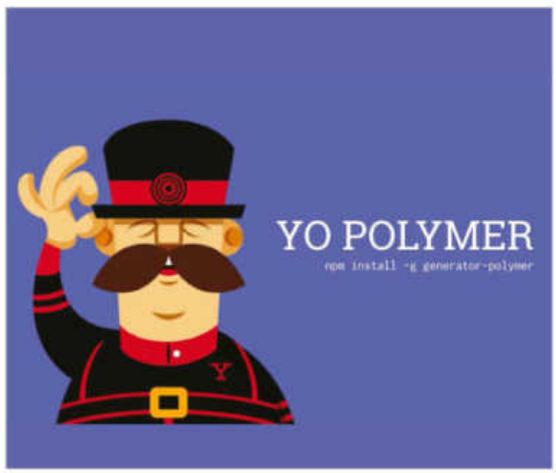
```
<link rel="import" href="../../bower_components/polymer/polymer.html">
<link rel="import" href="../../bower_components/paper-checkbox/paper-checkbox.html">
```

Next comes a `<dom-module>` element. This is used to tell Polymer where to look for your element's template. Make sure to give your `dom-module` an ID that matches the tag name you're creating – in this case it will be `todo-item`.

Every element's definition will begin with a list of imports at the top of the file

Inside the `<dom-module>` tag, create a `<style>` element followed by a `<template>`. These two form the UI for your element. Add the following markup to 'todo-item.html':

```
<dom-module id="todo-item">
  <style>
    :host {
      display: block;
    }
  </style>
  <template>
    <paper-checkbox checked="{{completed}}">
      <content></content>
```



Yo Polymer Power users will enjoy the speed at which they can build Polymer apps thanks to Yeoman and PSK

```
  </paper-checkbox>
 </template>
</dom-module>
```

The `:host` CSS selector indicates that this element should be `display: block` (instead of the default for all custom elements, which is `display: inline`). Within the template you have a `paper-checkbox` with a data binding on its `checked` attribute. This binding will enable you to toggle the state of the checked attribute based on data being passed into the component at runtime.

Finally, the `<content>` element allows anyone using our tag to pass in their own text content to act as the checkbox's label. For example, if someone were to write:

```
<todo-item>Buy some milk</todo-item>
```

On screen it would render a checkbox, followed by the words 'Buy some milk'. This is known as distribution and it's one of the cooler features of web components.

Now you need to give your element a prototype. Add a `<script>` tag after the `<template>` with the following content:

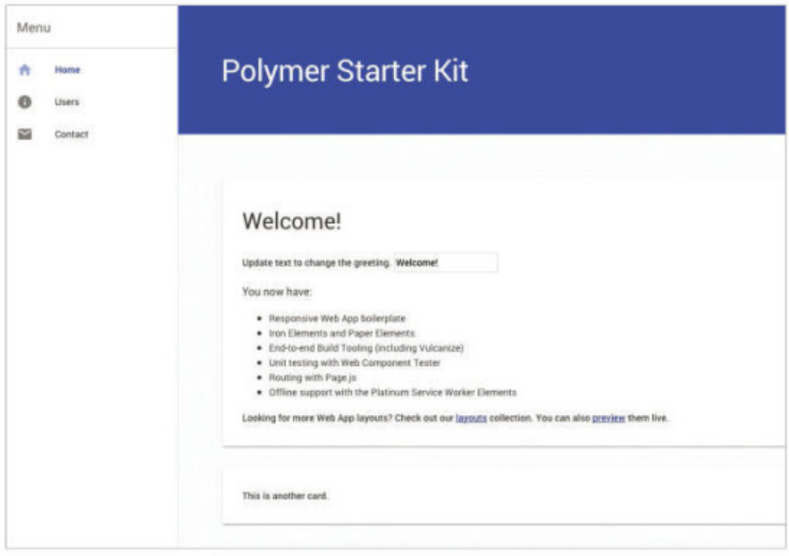
```
Polymer({
 is: 'todo-item',
 properties: {
  completed: {
   type: Boolean,
   value: false,
   notify: true
  }
 }
});
```

The `is` property tells Polymer what tag name to use when it registers the element with the document. You want to make sure it matches the ID you used on your `<dom-module>`.

The properties object (as the name implies) is used to define properties on your element. Here I'm specifying one property, named `completed`, and telling Polymer that it's a Boolean, its default value is false, and it should notify the outside world when it changes.

What this means is that any time the `completed` property is updated, it will automatically dispatch a `completed-changed` event. Doing so makes this property two-way bindable, hence the following line in our template:

```
<paper-checkbox checked="{{completed}}">
```



Design scaffold Out of the box PSK gives you an adaptive, material design scaffold that looks great

This means the outside world may pass in a value for the completed property on this element. If it is true, the checkbox will be checked, and if it is false, the checkbox will be unchecked. Because the binding is two-way, if the completed state ever changes (in other words, if someone checks the checkbox), the original data provider will be notified.

PODCAST

Make sure you tune in to Polycasts, a bi-weekly YouTube series dedicated to teaching Polymer and Polymer Starter Kit: *netm.ag/polycasts-271*

CREATE A TO-DO LIST

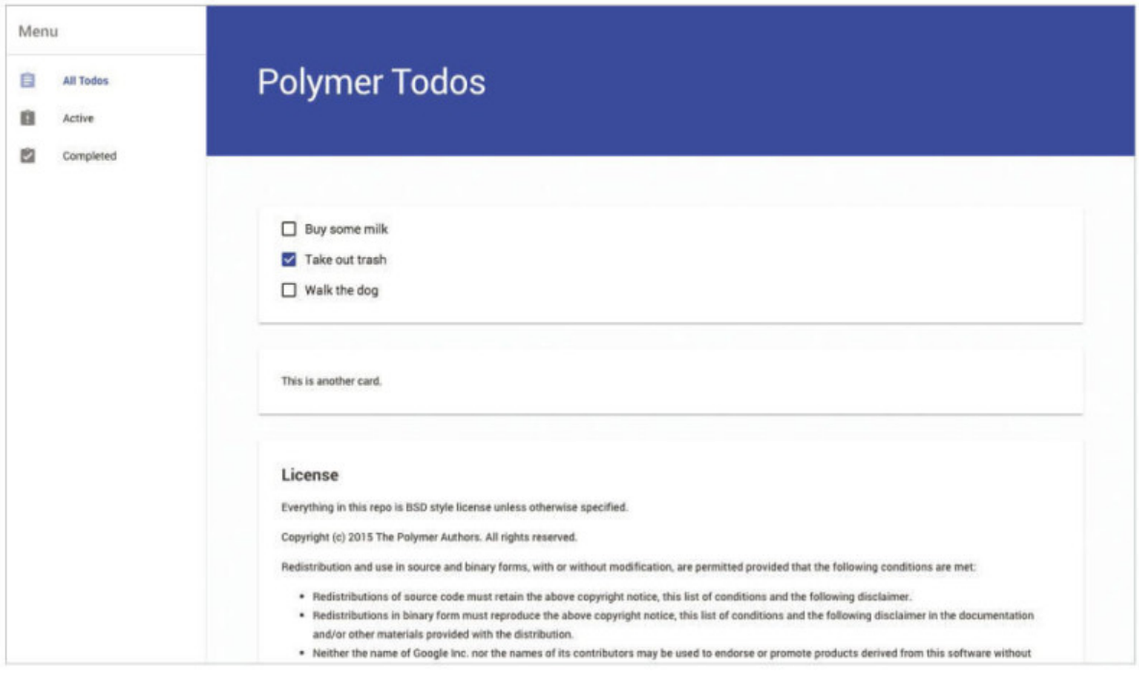
Now we have an element for to-do items, we can batch them together into a list. In the 'app/elements' directory create a new folder called 'todo-list' with a 'todo-list.html' file inside of it. At the top of the file, include imports for `polymer.html` as well as `todo-item.html`:

```
<link rel="import" href="../../bower_components/polymer/
polymer.html">
<link rel="import" href="../todo-item/todo-item.html">
```

Then create a `<dom-module>` with a `<template>` inside. Within the `<template>` you'll nest another `<template>`, but this time give it the attribute `is="dom-repeat"`. This signifies the element is a type extension custom element, so rather than just a plain old template, it's supercharged with the power of Polymer.

You can use a `dom-repeat` template to iterate over a collection and stamp out elements for each item it finds:

```
<template is="dom-repeat" items="{{todos}}">
 <todo-item completed="{{item.isComplete}}">
  <span>{{item.title}}</span>
```

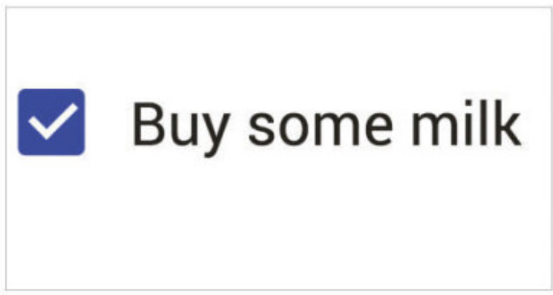


Just the beginning The final todo list. It's a great start, but there's plenty more we can do!

```
</todo-item>
</template>
```

Normally you would want the `todos` collection to come from an external data provider, but for the sake of brevity we'll hard-code it into our `todo-list`. Similar to the `todo-item`, you'll need to create a `<script>` tag that calls the Polymer constructor, passing in a tag name and `properties` object. Add the following lines to your `properties` object:

```
properties: {
 todos: {
  type: Array,
  value: function() {
   return [
    {title: 'Buy some milk', isComplete: false},
    {title: 'Take out trash', isComplete: true},
    {title: 'Walk the dog', isComplete: false}
   ];
  }
 }
}
```



Web components Content distribution is the secret sauce that really makes web components shine

To actually display the `todo-list`, open the 'app/elements/elements.html' file and add an import to the bottom.

```
<link rel="import" href="todo-list/todo-list.html">
```

Finally in the 'index.html' file, find the `<section data-route="home">` element and replace its content with `<todo-list>`. You should now see something that looks like the image above.

JUST GETTING STARTED

You've created a few Polymer elements, and your first material design app is shaping up, but there's so much more to do! Be sure to check out the GitHub repo accompanying this article (*netm.ag/polymergit-271*) for a more complete example, including how to add and remove to-dos.

Finally, if you've built something cool with Polymer, or just want to ask questions, join the Polymer Slack channel at *bit.ly/polymerslack*. I hope you enjoyed this lesson. Have fun hacking on Polymer!

VIDEO

Be sure to watch the Polymer session from Google I/O to learn all about Polymer 1.0: *netm.ag/google-271*

ABOUT THE AUTHOR

CLARK WIMBERLY

w: *clarklab.com*

t: @clarklab

job: UX designer

areas of expertise: Design, development, and that sticky area where things meet

q: what's the strangest thing you've ever eaten?
a: Rattlesnake, it tasted like frog

HEAD TO HEAD

COMPASS VS BOURBON

Variables and mixins and styles, oh my! **Clark Wimberly** explores the differences (and similarities) between Compass and Bourbon

COMPASS	BOURBON
Compass (*compass-style.org*) is a CSS authoring framework. It's built on Ruby and maintained by Chris Eppstein, who's on the main Sass team. Installation is a simple command, to create a project. If GUI is more your style, there's a Compass app available.	Bourbon (*bourbon.io*) is a Sass mixin library. It's built purely in Sass by the team from Thoughtbot, who maintain not only Bourbon but a family of related packages. Installation basically involves grabbing the files and `@importing` them into your Sass project.
SETUP AND USAGE	
While Compass helps you write CSS, it's definitely a Ruby library. This enables users to tackle more complex operations, like creating sprites at compile or checking *CanIUse.com* for the latest capabilities.	Bourbon is a mixin library composed entirely of Sass code. This means Bourbon can run in a Ruby-free environment, with compilers like LibSass. Some would argue it's faster, but that's a question for another day.
GRIDS	
While Compass doesn't include a grid system per se, there's a great selection of third party solutions. Susy, Zen Grids and Mueller all offer highly flexible, responsive grids. Setup is simple, then you're right back into Compass.	Grids in Bourbon are handled by Neat, one of the sister libraries offered by Thoughtbot. It's a responsive system that works mostly with mixins (keeping your HTML semantic and clean), and everything is fairly customisable.
TYPOGRAPHY	
To manage type size, spacing and more, Compass includes a vertical rhythm, powered by your variables. It allows you to establish a baseline at the root of your build, then manipulate with mixins for things like line-heights and borders.	For typographic scale and scaffold styles, Bourbon relies on Bitters, yet another member of the Bourbon family. Bitters aims to provide a jump-start on projects, with a variable-based, modular scale for most basic elements.
COMPILING	
This might sound silly, but one of my favorite features of Compass is how simple it makes compiling. A single `compass --watch` command will start polling the files you've specified when creating the project.	Because Bourbon is a mixin-only library, it relies on the vanilla Sass compiling functionality. A slightly more complex command is required (or the use of some app to compile your Sass for you).

FACT FILE

GET APPY
If you don't like the command line, both libraries are available in a number of desktop apps that make things a point and click affair.

NEW TO BOTH?
If you'd like to tinker with either Compass or Bourbon, online tools like CodePen make it easy to get started. You're just a few clicks away from including either library.

VERDICT

Offering variables, mixins and more reusable patterns than you can shake a stick at, both Compass and Bourbon are more than suited for production work. Because Compass relies on Ruby, it can offer more complex functionality and logic (like file-creation during spriting). Bourbon, basically a giant Sass include, may appeal to a lighter, more portable build.

View source files here!

All the files you need for this tutorial can be found at *netm.ag/RWDgit-273*

ABOUT THE AUTHOR

ERIC PORTIS

w: *ericportis.com*

t: @etportis

job: Printer, Pressure Printing, Inc

areas of expertise: Frontend development and photography

q: what did you want to be when you grew up?
a: Professional Teenage Mutant Ninja Turtle illustrator

SRCSET

BUILD SITES WITH RESPONSIVE IMAGES

Eric Portis walks through the new markup designed to help devs build performant, responsive sites with content images

Progressively enhanced pages with responsive layouts transform themselves to meet myriad users' myriad needs. A thoughtfully built page serves up quality experiences to everything from feature phones to 5K screens, becoming as rich and complex, or as fast as lean, as its context dictates.

That's the theory, anyway. In practice, the number-one thing holding back an efficient, adaptable web has been content images. Images delivered via the `<img>` element have traditionally been limited to a single `src`. Fixed in resolution, content and format, these `src`s are weighing down the responsive web.

Thankfully, new markup allows us to send different sources to different devices. This doesn't just solve the biggest performance problem on the responsive web, it also opens up new avenues for experimentation in frontend engineering and visual design. Now that we've broken the tyranny of the single `src`, our images can respond to as many different contexts as we can imagine.

Let's dive in with an example. *A Comic Natural History of the Human Race*, penned by Henry Louis Stevens in 1851, was the first American book of caricatures and the first American book to feature chromolithographic printing. It's a weird, funny, and technically innovative book of images. Let's give it the web treatment it deserves (find a demo of what we're creating at *netm.ag/demo-273*).

WORKING WITH SRCSET

We'll start with the homepage, which contains fixed-width thumbnails of each illustration. We want these

VIDEO

Eric Portis has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/rwdvid-273*

Comic history Henry Louis Stephens' original *A Comic Natural History*

thumbnails to look crisp on hi-DPI (aka Retina) devices, and load as quickly as possible on standard screens. Until recently, we have had to pick one of these two contexts to optimise for. New markup allows us to have our cake and eat it too:

```
<img src="the-woodpecker/thumb.jpg"
  srcset="the-woodpecker/thumb@2x.jpg 2x"
  alt="A woodpecker with a man's head" />
```

We leave our `1x` resource in the `src` and render a separate `2x` resource, which we mark up using a new attribute: `srcset`. You may be wondering, why the `set` part? Because we can stick as many resources in there as we please, separated by commas:

Unlike with srcs, new markup allows us to send different sources to different devices

```
<img src="standard.jpg"
  srcset="retina.jpg 2x,
    super-retina.jpg 3x,
    wtf-retina.jpg 4x"
  alt="My god, it's full of pixels!" />
```

Those little `2x`s and `3x`s sort of look like little media queries, don't they? Perhaps they're a kind of shorthand? Nope! Media queries make true or false statements about the browsing environment and provide the browser with an explicit set of instructions. `srcset` descriptors simply describe resources. When we put a `2x` next to a resource, we're not giving the browser an instruction about when or whether to load it. We're simply stating it has an 'image density' of `2x`.

IN-DEPTH

BROWSER SUPPORT AND FALLBACKS

Thanks to the extraordinary efforts of developer Yoav Weiss, Blink-based browsers were the first to support responsive image features. Chrome (v39) and Opera (v30) began shipping with full support last autumn, and support trickled down into Chrome for Android soon afterwards. No version of the stock Android browser (which shipped with Androids <5) supports the markup.

Firefox

Firefox 38 began supporting the new syntax in May of this year. Mozilla's implementation contains one notable flaw: it doesn't yet re-evaluate responsive images in response to resize events. A patch has landed in the beta channel, and should ship in version 42 in November 2015.

Safari

Mobile and desktop Safari have supported `srcset` with `x` descriptors since version 7.1 and version 8.3 respectively; support for `w` descriptors will land this autumn after the release of Safari 9 and iOS 9. Apple have made no official statements about `picture` but there are positive signs.

Internet Explorer

No version of Internet Explorer supports the new markup, but Microsoft's just-released Edge browser supports `x` descriptors, and both `w` descriptors and the picture element are officially 'in development'.

Non-supporting browsers

All the new features build upon our familiar friend `<img>`. This ensures that non-supporting browsers will simply ignore the new stuff and load the `<img src>`, same as they always have. If, however, you want to polyfill support for the new features, you're in luck: Picturefill.js is a superb, actively maintained, thoroughly-tested 6Kb library that'll do exactly that.

It's not without its gotchas though – because browsers start loading images before they run any JavaScript, when using Picturefill you'll have to either omit `src`s or risk burdening users with double downloads. Find out more at *netm.ag/picturefill-273*.

FOCUS ON

THE MEDIA ATTRIBUTE

On the Comic Natural History site, we use `picture` and `source` for their type-switching prowess. They have another trick up their sleeve, though: the `media` attribute.

```
<picture>
  <source media="(min-width: 20em)"
    srcset="full-image.jpg" />
  <img src="cropped.jpg"
    alt="Sometimes, less is more" />
</picture>
```

`srcset` lets us supply the same image at a range of resolutions, and gives the browser the leeway to choose whichever resource it wants. But sometimes we'll want to supply resources with alternate content, and make sure that those alternates are always loaded in certain browsing environments – no ifs, ands or buts about it.

For instance: maybe we want to supply a black-and-white version of an image for printing, or a cropped version for small screens. That's called art direction, and that's what the `media` attribute is for. `<source media>` works just like `<source type>` – the first `<source>` that has a media attribute that matches the current environment wins, and its `srcset` is sent to the `<picture>`'s `<img>`.

Today, `media` attributes are mostly being used to send alternate crops, but I believe we've only just begun to explore what can be done with them. In 'When Responsive Images Get Ugly' (*netm.ag/ugly-273*) Taylor Hunt uses `media` to replace animated GIFs with poster images on ereaders whose slow screens turn the GIFs to mush. In another example, he alters images based on the device's ambient light sensors, so that a floor plan remains legible and easy-to-read in both bright sunlight and at night. Exciting stuff!

Focus in Alternate crops enable users to focus on the important part of an image at any screen size

What's an image density? I'm glad you asked! It's the number of pixels a resource has, relative to its layout size. More precisely: Resource width (in pixels) ÷ the `<img>`'s width on the layout (in CSS px) = image density (e.g. 2x). So if we have a 640×480 resource and we cram it into a 320px-wide box on our layout, it has a density of 640 ÷ 320 = 2x.

We can only know an image's density when we know its layout dimensions. And we can only know an image's layout dimensions when they are fixed. If an image is fluid and styled relative to its container (with, say, a `max-width: 100%` rule), simple `1x` / `2x` descriptors won't suffice.

W DESCRIPTORS

This brings us back to our example. The illustrations on our detail pages are fluid. Once the viewport shrinks below 640px wide, they scale down along with it. We deal with this variability using new markup to split image density into its two component parts: resource width and layout width.

We tell the browser the width of our resources using the `w` descriptor. And we'll specify the `img`'s layout width using the `sizes` attribute.

```
<img src="medium.jpg"
  srcset="small.jpg 320w,
    medium.jpg 640w,
    large.jpg 1280w"
  sizes="100vw"
  alt="A woodpecker with a man's head" />
```

`w` descriptors are easy. How wide is the resource? Put a `w` after that number and you're done. 'small.jpg' is 320×431, so we mark it up as 320w; 'medium.jpg' is 640×862, so it gets a 640w, and so on.

SIZES

`sizes` answers a different question: how wide will the image be on the layout? `sizes` takes CSS lengths, like `10em`, `500px`, or (if you fancy) `calc(80vw - 30px)`. In the previous example, I've used a `sizes` value of `100vw`, stating that the image will occupy 100 per cent of the width of the viewport.

What do browsers do, upon encountering this markup? They convert the `sizes` value to CSS px and work out the image density of each resource in the `srcset` accordingly. On a 320px-wide viewport, our 640w source would have an image density of 640w ÷ 320px = 2x. But on a 480px-wide viewport, that same image would only have 640w ÷ 480px = 1.333x.

Browsers determine these image densities on the fly. And then they do something wonderful: they pick whichever resource they want to! Usually 1x screens will get something like a 1x resource and 2x screens

will get something bigger. But browsers are not bound by any particular picking logic. If a browser knows it's running on a slow connection, or perhaps if a larger-than-necessary resource is already in the cache, that browser is free to choose a different resource than it might have otherwise.

Back to our example. `100vw` doesn't quite capture the dynamic width of the illustrations on our detail pages. They have a `width` of `100%`, but the `figures` that house them have a `max-width` of `640px`. How do we tell the browser that?

`sizes` lets us supply multiple values in a comma-separated list and tell the browser which length to use, based on media conditions. If the layout width of our image changes once the viewport reaches a certain size, we can encode that information in `sizes`, like so:

```
<img src="medium.jpg"
  srcset="small.jpg 320w,
    medium.jpg 640w,
    large.jpg 1280w"
  sizes="(min-width 640px) 640px, 100vw"
  alt="A woodpecker with a man's head" />
```

Sizes lets us supply multiple values in a list and tell the browser which length to use

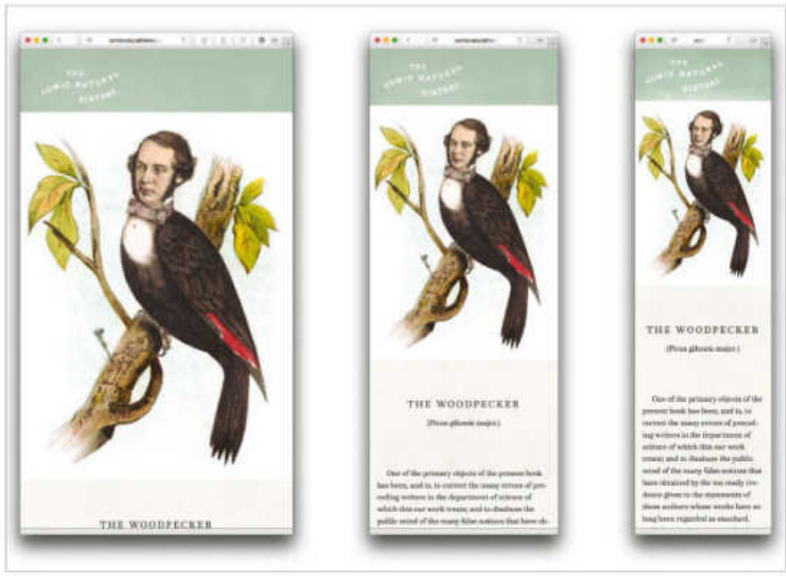
This says: if the viewport is at least 640px wide, the image will be 640px wide. If it's not, the image will be 100vw wide. Our `sizes` now conveys our image's dynamic layout width exactly.

PROGRESSIVE ENHANCEMENT

And with that, we've marked up all our images so they'll scale efficiently in response to both variable screen densities and their own dynamic layout widths. There is, however, another dimension that we can adapt across: format support.

Let's say we want our images – currently trapped within jarringly bright white boxes – to appear as if they've been drawn directly onto the pages' paper backgrounds. To achieve that effect, we need to encode them with a format that supports transparency.

PNG does, but it isn't very good at compressing images with a lot of texture and subtle gradations – if we used PNG, the file-sizes would be prohibitively large. What's a modern web developer to do? I'll tell you what: progressively enhance!



Size down The illustrations on our chapter pages shrink along with the viewport

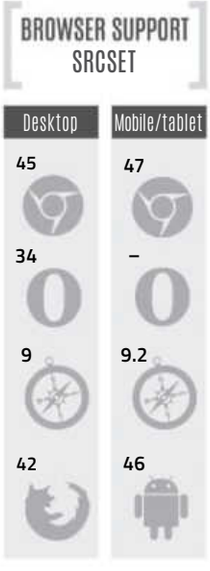
Google's WebP format compresses our images well *and* supports transparency, but it's only supported in Blink-based browsers like Chrome. That's OK by us. We'll use new another new markup pattern (borrowed from `<audio>` and `<video>`) to send WebPs to browsers that support them, and JPGs to everyone else:

```
<picture>
  <source type="image/webp"
      srcset="small.webp 320w,
        medium.webp 640w,
        large.webp 1280w"
      sizes="(min-width 640px) 640px, 100vw" />
  <img src="medium.jpg"
      srcset="small.jpg 320w,
        medium.jpg 640w,
        large.jpg 1280w"
      sizes="(min-width 640px) 640px, 100vw"
      alt="A woodpecker with a man's head" />
</picture>
```

Browsers send the `srcset` of the first `<source>` whose type they support to the `<img>`. Thus, the (required) `<img>` is still the thing that we see on the page. We're progressively enhancing it by wrapping it within a `<picture>`.

RESPONSIVE RESULTS

Our images now adapt across three dimensions: screen density, viewport width and format support. We've broken the tyranny of the single `src` in order to mark up images that are as responsive as the pages that house them, taking advantage of high-resolution screens and cutting-edge formats without sacrificing performance for those less capable. Huzzah!



ABOUT THE AUTHOR

MATT GRIFFIN

w: *bearded.com*

t: @elefontpress

job: Founder, Bearded

areas of expertise: Responsive web design, frontend development

ABOUT THE AUTHOR

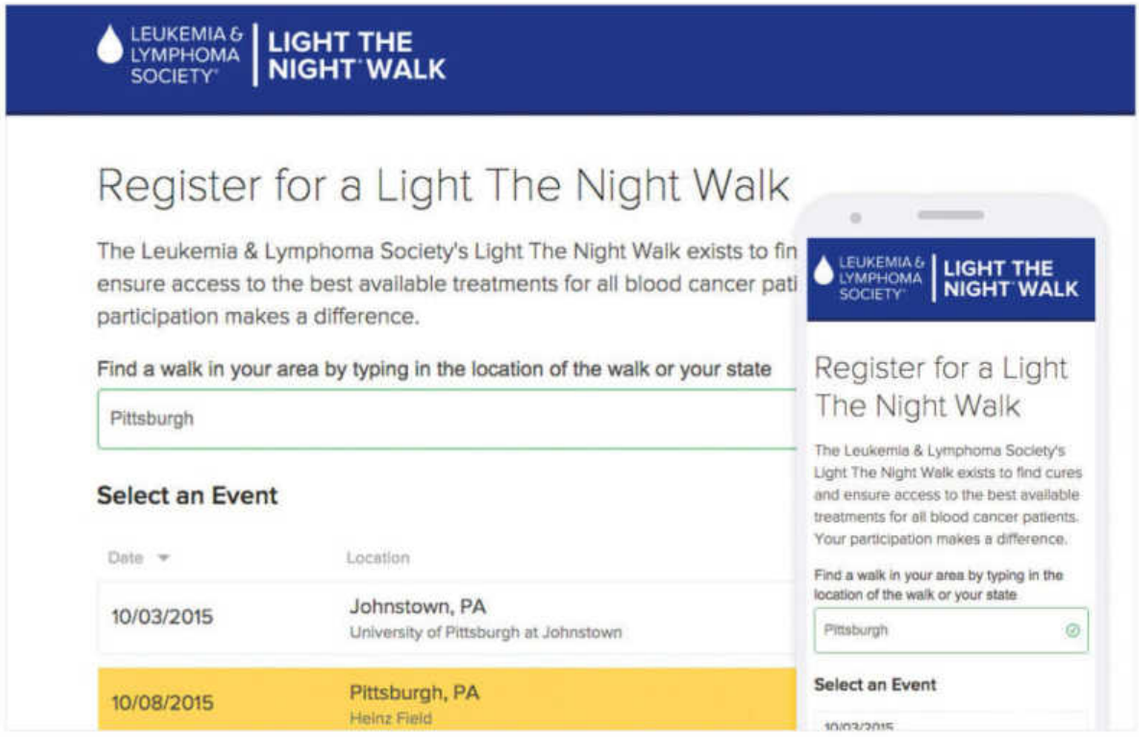
PATRICK FULTON

w: *bearded.com*

t: @patrickfulton

job: Developer, Bearded

areas of expertise: Responsive web design, frontend development



FORMS & TABLES

CREATE RESPONSIVE FORMS AND TABLES

Matt Griffin and **Patrick Fulton** tackle two of the most challenging design elements in RWD: forms and tables

Some years ago we started a small web agency called Bearded. In 2010 responsive web design completely changed how we work. Ethan Marcotte's simple yet groundbreaking approach pointed the way towards a web that embraces the constantly expanding world of mobile devices, ensures content parity, and takes progressive enhancement to its next logical step. Responsive web design directly responded to the problems many of us had already felt in web design: it had become too brittle, too precious and too pixel-specific. The web wants to be fluid, and it needed a fluid solution. And that's what we got.

Now it's 2015: about five years after RWD started being practiced commercially. We've made a whole bunch of responsive sites during that period. And you know what? It's still hard. Making the mental shift from a fixed-width world to a flexible one requires us to rethink many design patterns and user interface conventions.

Not surprisingly, the hardest parts of RWD are things that have always been kind of a pain: tables and forms. Let's have a look at why they are even more fun in a responsive environment.

TABLES

The first rule of tables? Don't use tables! But seriously, let's think about this for a second. I like to think of tables as kind of a pre-standards-conscious proto-HTML. Tables, by their nature, attempt to do

RESOURCE

The folks at Zurb have released their own CSS/JS solution to responsive tables, which plays well with Foundation and non-Foundation-based sites: *netm.ag/zurb-272*

visual layout with HTML. Their rows and columns and endless table data cells break the cardinal rule of the post-web standards project world: the separation of style and content.

So before you start adding table tags to your markup, stop and ask yourself: is this tabular data? Tabular data derives its meaning from the comparison of similar facets of an information set. In other words, if the value in the information comes from comparing data across columns and rows. If this is the case, you may need a table.

Things that are not tables

If the data has significant value unto itself, maybe it doesn't need to be a table. Or at any rate, not always. Some other options to keep your content out of tables? Headings and paragraphs, or bulleted lists where each item begins with a bold inline heading.

Tables break the cardinal rule of web standards: separate style and content

Things that appear to be tables

A good example of something that may initially look like a table, but is not actually tabular data? A calendar! A calendar seems tabular because the columns share a common set of designations: Monday, Tuesday, Wednesday, and so on. It is a convenience that we can identify all the Wednesdays at once. This is why people use calendars.

However, the data you put on calendars (say a bunch of Broadway musical performances) is not intrinsically tabular. Comparing one day's shows to another's does not necessarily improve users' understanding of the performances on individual days. A calendar is one way to display date-organised data, but there are others too.

So what does a calendar look like on a small screen? We're asking the wrong question. The right question is: what does a list of events look like on a small screen?

This is how we arrived at marking up our days and events using `<divs>` and `<articles>` rather than tables on the Children's Museum of Pittsburgh website (*pittsburghkids.org/calendar*). It's much easier (and more semantically appropriate) to make a bunch of `articles` and `divs` look like a table, than to make a table look like a list. ▶

FOCUS ON

PRESENTATIONAL CLASS NAMES

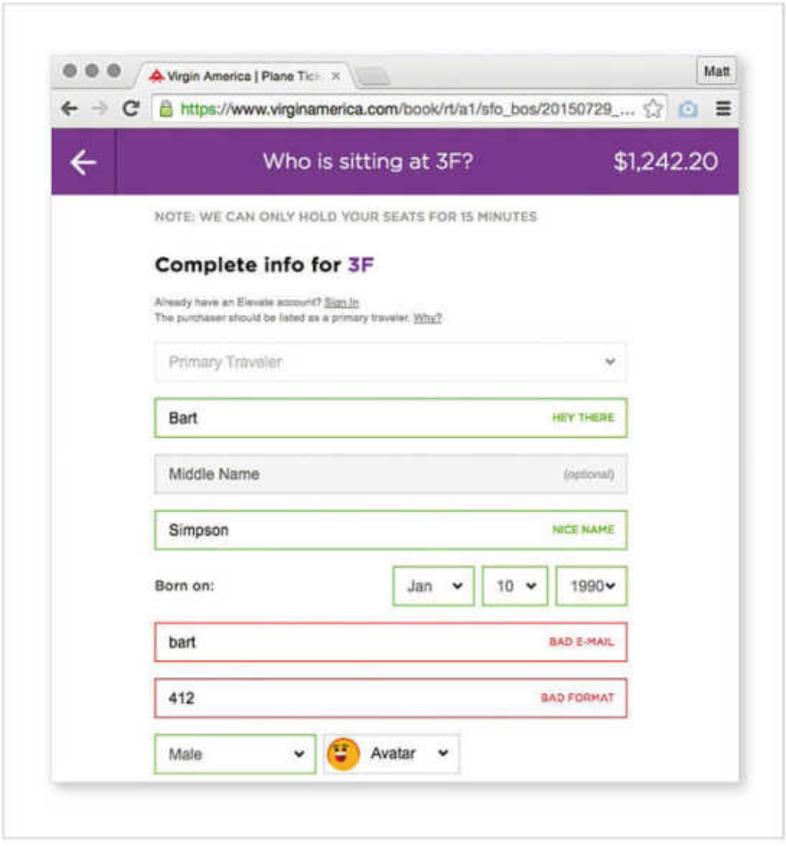
In general, we always strive for semantic class names. However, when dealing with large-scale, pattern-driven design and development, there are times when presentational naming makes more sense.

For instance, for form input layouts, we may need to employ some extra markup whose only purpose is presentational, to enable us to lay out elements on a grid. For this, the most semantically appropriate names are presentational ones. Let's look at the HTML (full example at *netm.ag/box1-272*):

```
<div class="two-up-wider">
 <div class="input">
  <label for="firstname">First Name</label>
  <input id="firstname" class="valid icon-valid" type="text"
value="Lisa" name="firstname" />
 </div>
 <div class="input">
  <label for="lastname">Last Name </label>
  <input id="lastname" class="valid icon-valid" type="text"
value="Simpson" name="lastname" />
 </div>
</div>

<div class="two-up-wider">
 <div class="input">
  <label for="address1">Address</label>
  <input id="address1" class="invalid" type="text"
name="address1" />
  <span class="validation-text invalid">invalid</span>
 </div>
 <div class="input">
  <label for="address2">Apt. / Suite / Unit
   <span class="optional">(optional)</span>
  </label>
  <input id="address2" type="text" name="address2" />
 </div>
</div>
```

In this case, we're giving the container `<div>`s around our sets of inputs classes like `"two-up"` and `"two-up-wider"`. Each set of labels and inputs within is wrapped in a `<div>` with the class `"input"`. This allows us to easily and programmatically style our forms simply by using consistent layout-based classes. Give a set of inputs the right class, and they behave accordingly.



Helpful and personal Virgin America's website offers helpful and succinct form validation messages with great personality

Things that must be tables

As much as I hate to admit it, some information *is* tabular. When designing the fundraising event registration process for the Leukemia Lymphoma Society (*registration.lightthenight.org*), we used tables for event selection. Quickly scanning and sorting this list by location and date in order to choose the particular event you want to register for seems like a good use case for tables. But how to make them responsive?

Our investigations into responsive tables started with Chris Coyier's excellent April 2011 responsive data tables roundup: *netm.ag/coyier-272*. For several projects in the past few years we'd consult this article and inevitably, Chris' first approach (*netm.ag/winner-272*) was the winner.

On wide screens, you get tables. On smaller screens, you get each entry with all of the header labels restated for context. You lose the comparative feature of tables on small screens, but you do get the essential information in an easily readable way. In other words, we treat the tabular layout as a progressive enhancement. This is our favourite UI pattern for responsive data tables, and it seems like our friends at Filament Group agree with us. In August 2013 the Filaments released Tablesaw (*github.com/filamentgroup/tablesaw*) on GitHub, which includes this pattern as one of its three options. Tablesaw is the starting point for the LLS example, and I highly recommend using it as a jumping-off point for creating your own responsive data tables.

So if Tablesaw works so well, why wouldn't you just use that all the time and forget about it? Though some major interface or display tables (like the event picker example, or maybe an SaaS pricing matrix (*smallfarmcentral.com/plans*)) make sense as custom HTML, what about when a site content manager wants to toss a table onto a page using their CMS?

If site managers are entering tables in the CMS using a WYSIWYG editor, there are some issues to surmount. First of all, Tablesaw does its magic by looking at the `<thead>` and using what it finds there to label the information further down in the `<td>`s on smaller viewport widths. If the site manager isn't entering `<thead>` information properly, things won't work.

It's also important to note that Tablesaw only acts on tables with the right classes applied to them. You now need to decide if all tables are Tablesaw tables, or if that's an option in the CMS controlled by the site manager, then apply those rules in your CMS so it outputs the right markup.

Additionally, there are some display issues to consider. Repeating the column headers in every cell tends to add a fair bit of height to the table. It's not a very efficient way to display the data. A non-tabular approach using headings, lists and paragraphs could be an easier read on mobile, depending on the content.

This is why my first approach to tables is still 'don't use tables'. Tablesaw should be your safety net, not your go-to tool. But believe me, you'll be glad it's there when you need it.

FORMS

It's likely tables have been a thorn in your side on projects before. But their persnickety nature is nothing compared to our other good high-maintenance HTML friend: forms.

Because forms are so browser and device-specific in terms of how they react to styling, they can be a real challenge to work with. To paraphrase Luke Wroblewski, forms drive one of the most essential aspects of the web: the ability to generate revenue. Yet filling out a form is the last thing anyone wants to do on a website. Thus, forms make for a fun set of design problems. Let's have a look, shall we?

LAYOUT

When it comes to laying out forms, there are a few recurring patterns we use. For consistently short pairs of inputs, we use a pattern that we call two-up. It's pretty straightforward. No matter the size of the viewport, these inputs appear side-by-side. For inputs that need a little more space, but can be next to each other when they have enough room, we use the pattern two-up wider. There are rarely situations where three-up – three side-by-side inputs – always works. So by default, we have that work on the wider principle.

One special case for three-up is postal code, city and state. Because of the nature of these fields, custom unequal widths better serve their purpose. Their styling works on the same principals as before, but with a bit more customisation for the unequal widths we want to achieve with postal code, city and state combos.

You might have also noticed that we're starting the cluster with the postal code field. Both Luke Wroblewski and Brad Frost have shared a great approach where the postal code comes first (*netm.ag/postcode-272*), allowing us to autofill state and city, which is much more efficient for users.

With imput messages, brevity is important. The most brief response possible? Icons!

VALIDATION

No matter how good our forms are, sometimes users will have a tough time filling them out. This is where good validation comes into play. For input validations, we like to consider two components: messages inside the input and messages outside the input.

Inside the input

The thing to consider with messages inside the input is space. You can't, for instance, prompt a user with password requirements inside an input. But there is likely enough room for 'invalid' – and there is definitely enough room for a happy green checkmark when they get it right.

One of my favourite parts of Work & Co's Virgin America's responsive redesign (*virginamerica.com*) is how it deals with form validation. It's good at communicating a helpful validation message succinctly (e.g. 'bad email').

With responsive and small screens, brevity is important. The most brief response possible? Icons! And we designers do love our icons. The only problem is that most icons aren't universally understood. For success with a form field, that green checkmark is probably fine. But how should we indicate failure? A red 'X'? But 'X' in an interface means close or collapse. The word 'invalid' is OK, but a whole lot longer than most icons.

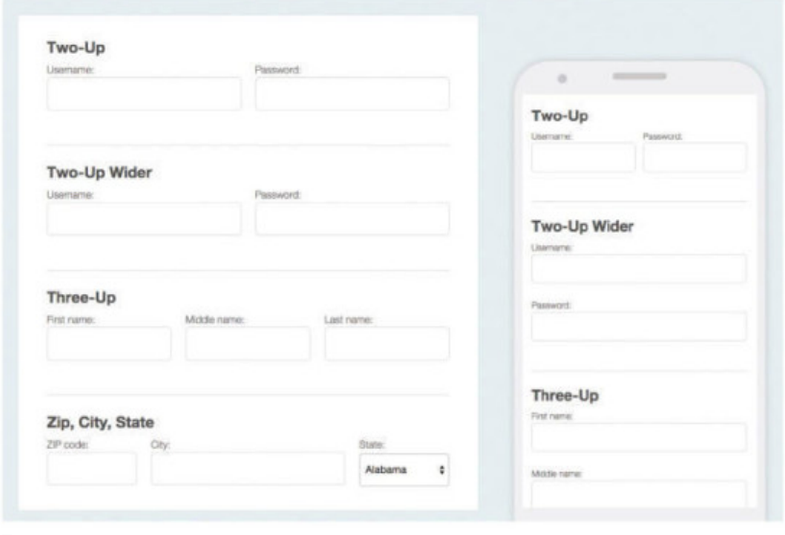
If the tone and personality allows it, a frowny face seems relatively clear. But many brands aren't as playful as Virgin America. In our work at Bearded we've settled on an appropriation of the 'do not enter' sign. Its similarity to iOS's 'remove item' icon isn't ideal, but it's the best solution we've come up with so far.

One thing about responsive design and progressive enhancement, though: we can implement an icon where space is at a premium, and expand to an inline validation message like 'invalid' or 'bad email' when we can afford the extra width.

Outside the input

When we need to display feedback to a user beyond what we can cram inside an input (or when elements like selects are involved), it's time to start using messages outside the input.

In those circumstances, you can use both kinds of validation. For example, you can add an inline 'invalid' message, then below the input tell users the longer story. In this space, you even have the luxury of providing links to enable users to perform alternative actions (e.g. 'Sorry this email address is already used with an account. Try signing in').



Two-up At Bearded, we use certain responsive layout patterns over and over again for form inputs

RESOURCES

FURTHER READING

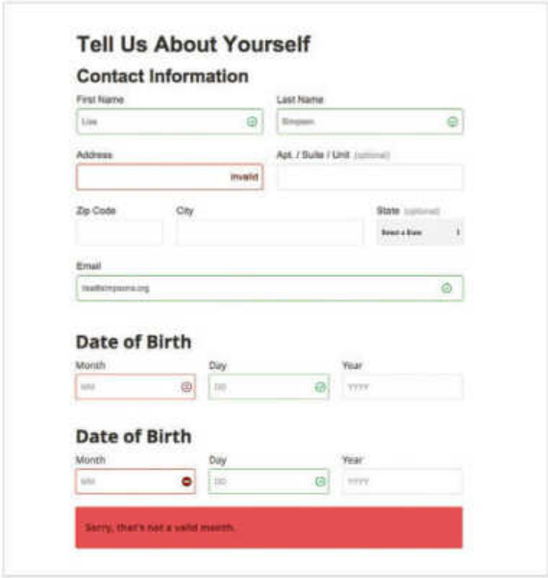
Forms and tables are hard. Luckily, some very smart people have written down their thoughts to share with you. Here are a few of our favourite examples:

Web Form Design
rosenfeldmedia.com/books/web-form-design
Luke Wroblewski's book belongs in every web designer's library. In it, he takes an analytical look at what may be the web's most important and vexing component, offering real world examples and a variety of solutions to common problems. If you don't have it, buy and read it now.

Tablesaw: A Flexible Tool for Responsive Tables
filamentgroup.com/lab/tablesaw.html
The Filaments have given us a lot of great solutions over the years, but Tablesaw is one of the ones we use the most often – it gives you three very good options for displaying data tables in a responsive manner. Mat Marquis and Zach Leatherman give you a rundown in this helpful blog post.

Dropdowns should be the UI of Last Resort
lukew.com/ff/entry.asp?1950
When you talk about web forms, that Luke Wroblewski really seems to come up a lot, doesn't he? His quick but insightful look at selects is great at reminding us that plenty of modern UI options out there are superior to the old drop-down.

Why and how we ditched the good old select element
netm.ag/select-272
Mikkel Bo Schmidt lays out Tradeshift's travails with the select element, which ultimately led the company to ditch it from its product altogether. This article was instrumental in helping us down a path that ultimately arrived at our prioritised selector design. Definitely worth a read!



Form validation For our work on the Leukemia Lymphoma Society websites, we tried a number of approaches for form validation

Regardless of your approach, form interactions are a subtle art. I would encourage you to run usability testing on your prototypes as you go. Spending a week trying things out with five users can give you a wealth of information about how things are (or are not) working, before you go live.

DATES

One of our favourite input use cases? Date selection. Why? Because it's so hard to do well! Let's say users need to select a date to make an appointment in the near future. Some things to consider:

Date.js allows users to enter dates however they want. Slashes, dots, spelled out ...

- Users must be able to input the date in a format the database on the other side will be happy with
- Users need to know which day of the week a date is as they consider their schedule

Sounds like a calendar picker is the way to go. But calendars on small viewport are the worst, right? Luckily pickadate.js (*amsul.ca/pickadate*.js) does a good job of handling this problem. With a few UI customisations, your calendar picker is good to go.

That's great for near-future date selection. But what about selections in the distant past – for instance, birth date? You'd have to click your way back decades using the previous month button. How odious! Thankfully, for this sort of interaction day of the week is irrelevant, so the only real challenge is formatting. So what to do? You could try to explain to users what to do using hint text. Except that, well, no one reads hint text.

OK, so we can force formatting choices with three select inputs – one each for month, day and year. That day select input with its 31 options is no picnic – but it's a dream compared to the year select going back to 1915.

Fear not, for natural date entry is a problem that's been solved. Date.js (*netm.ag/date-272*) allows users to enter dates however they want. Slashes, dots, or hyphens; abbreviated or spelled out. What's more, it confirms the output with you using a handy validation message, and tells you day of the week, for kicks.

So … dates? Feels good man. But let's roll back to selects for a second.

SELECTS

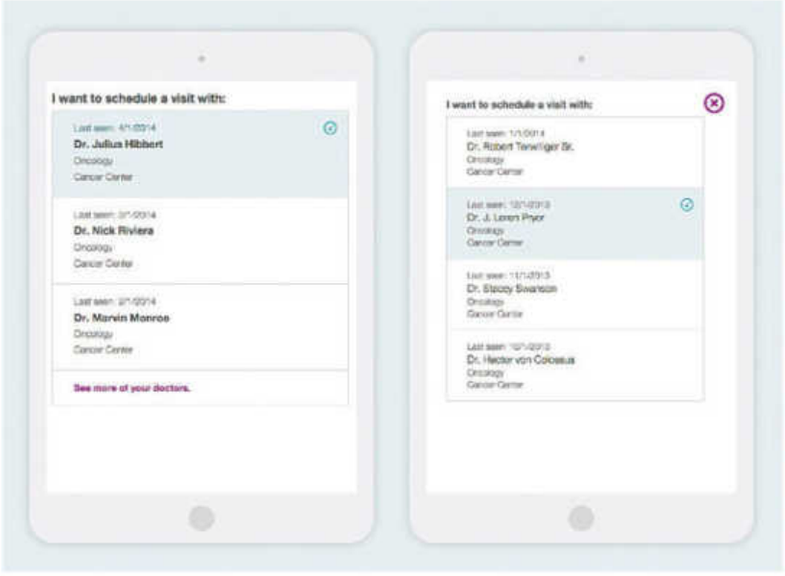
Even for form inputs, selects are kind of a bugbear. Styling those son of a guns consistently across browsers is nigh on impossible. Chris Coyier has put together a great write-up of the common pitfalls and limitations (*netm.ag/tricks-272*). At Bearded, our standard approach to selects is to match up their height to other inputs, but otherwise pretty much leave them alone.

It's worth noting that the Filament Group has made a heroic attempt at cross-browser select styling, available on GitHub (*netm.ag/cross-272*). However, although I love the idea of gaining styling control over these little troublemakers, my personal feeling is that this kind of approach is a bit on the brittle side.

Long selects

The worst situation for selects? When they get long! The fine folks at Harvest were nice enough to release some of their custom select approaches to help you with this, in the form of jQuery plugin Chosen (*harvesthq.github.io/chosen*). One of these approaches provides a searchable select that helps users reduce the number of options in front of them. And thanks to Nathaniel Flick, they've been responsivised: *netm.ag/responsiveselect-272*.

Chosen has some super-fun features, but it can, like any select-replacement approach, lead you down some rabbit holes. Be careful not to make things more complicated than you have to.



Complex selectors This complex select UI allows users quick access to prioritised options (left), with modal access to additional options (right)

Complex selects

Recently we worked on a project for a healthcare provider in which users needed to select from complex groupings of information. There was a list of doctors, their names, the doctor's area of specialisation, their office location, and the date they last saw the patient.

Complex information like this will not fit into a standard select element, so we tried something else. The solution we came up with is something we called the prioritised selector. In it, we show the user the first few options, prioritised by some specific criteria (most recently visited, for example), as well as the option to view more.

When a user selects the 'view more' option, we open a full-screen modal interface that lets the user select from the full list. Though it adds an extra click when a user needs to access doctors they haven't seen in a while, this ultimately seemed like the best approach for the use cases we were accounting for.

LET'S FACE IT: THESE THINGS ARE MESSY

Tables and forms are complex by nature. No matter how much we work on them, they might never be easy. And as the environment we deploy them into grows ever more complex, so will the design problems they represent.

But isn't that the fun of web design? As we solve the problems that are there to be solved (and thus the threat of boredom and complacency rears its ugly head), the web always seems ready to mix it up again, keeping us on our toes. And thank goodness for that.

RESOURCE

Zoe Gillenwater's excellent 'Enhancing Responsiveness with Flexbox' presentation explains how flexbox can help enhance the layout of your forms: *netm.ag/gillenwater-272*

Written by the industry's leading experts, The Ultimate Guide to WordPress Vol II shows you how to get the best from the world's most popular content management stystem

Pick up your copy at

netm.ag/WordPressguide

FOCUS ON

RESPONSIVE FORMS

Making forms responsive may be simple, but paying attention to the device can make a big difference to usability, says **Gene Crawford**

Making your web forms responsive is not a topic that's discussed too often. Mostly because it's not a huge deal to get HTML form elements to behave in a responsive manner – that is, to be both liquid and to utilise targeted screen widths with media queries to serve up specific form layout changes.

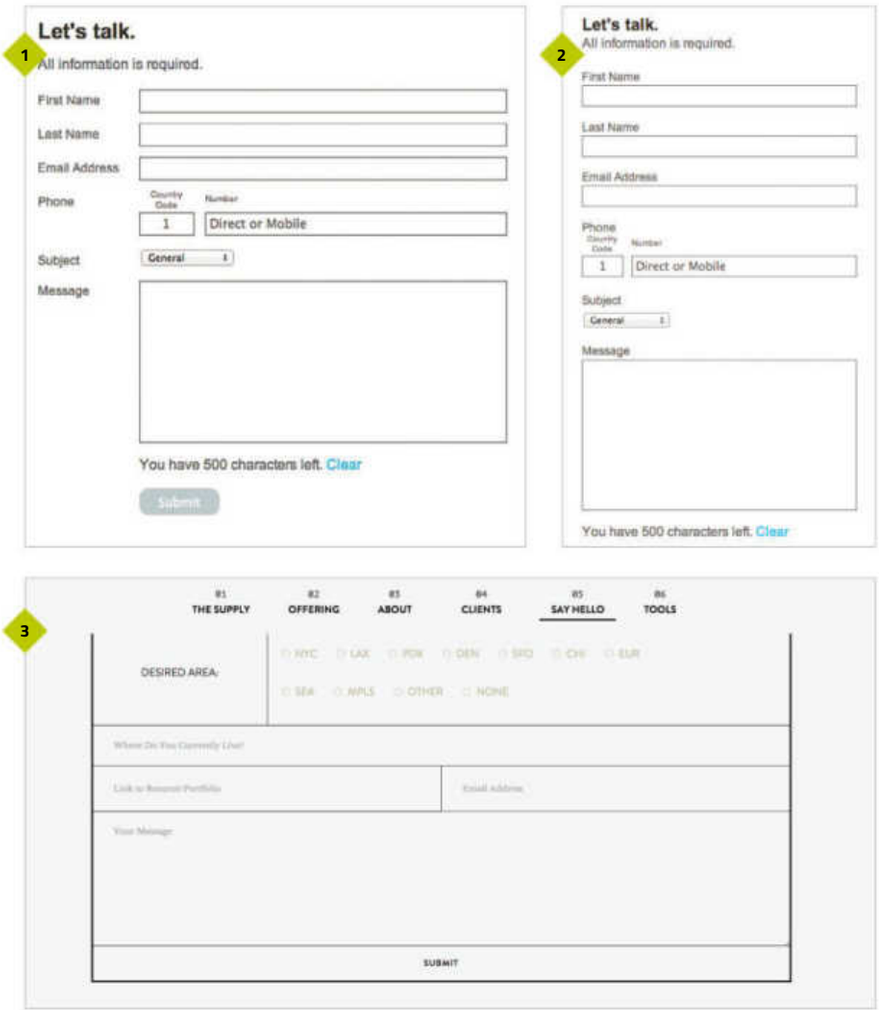
The main thing is to consider the context in which a person will be using the form. This a question of thinking through the device you are targeting, not just its screen size. Let's look at a couple of examples that help illustrate my point.

First, field sizing is important on smaller screens when people are using their fingers to target a form field. When using our phone, we do not have the same level of precision as we do with a mouse. Apple's Human Computer Interaction Design documentation (*netm.ag/doc-271*) has some great in-depth information on how to design for people who are using their fingers to interact with a device.

Another consideration when designing a responsive layout is the placement of form elements in relation to their respective fields. On desktops and iPads, having the labels left-aligned makes the form appear simple to complete. If we change the same form on an iPhone's screen so the labels are on top of the fields, this makes the user slow down a little. As mobile users often have their attention split between different tasks, this should help focus their efforts. Accuracy is more important than speed.

PROFILE

Gene's mission is to work tirelessly to provide inspiration and insight for developers. His recent projects include *unmatchedstyle.com*



(**1**) Here's a great example of a typical desktop browser-based contact form: the MCD Partners website (*mcdpartners.com/contact*). Notice the field titles are left-aligned on this version of the form. (**2**) This is the same form for the MCD Partners website, but on an iPhone. Here, the field titles are stacked vertically to save space and make things easier for the user on the long scrolling page. (**3**) The main form on the talent recruiting company The Supply's website (*thesupply.com*) uses inline form field labels, to create much the same effect as the other examples.

View source files here!

All the files you need for this tutorial can be found at *netm.ag/type-267*

ABOUT THE AUTHOR

CLARISSA PETERSON

w: *clarissapeterson.com*

t: @clarissa

job: UX designer and web developer, Peterson/Kandy

areas of expertise: RWD, UX design, content strategy

q: what's your favourite sweet treat?
a: Chocolate cupcakes

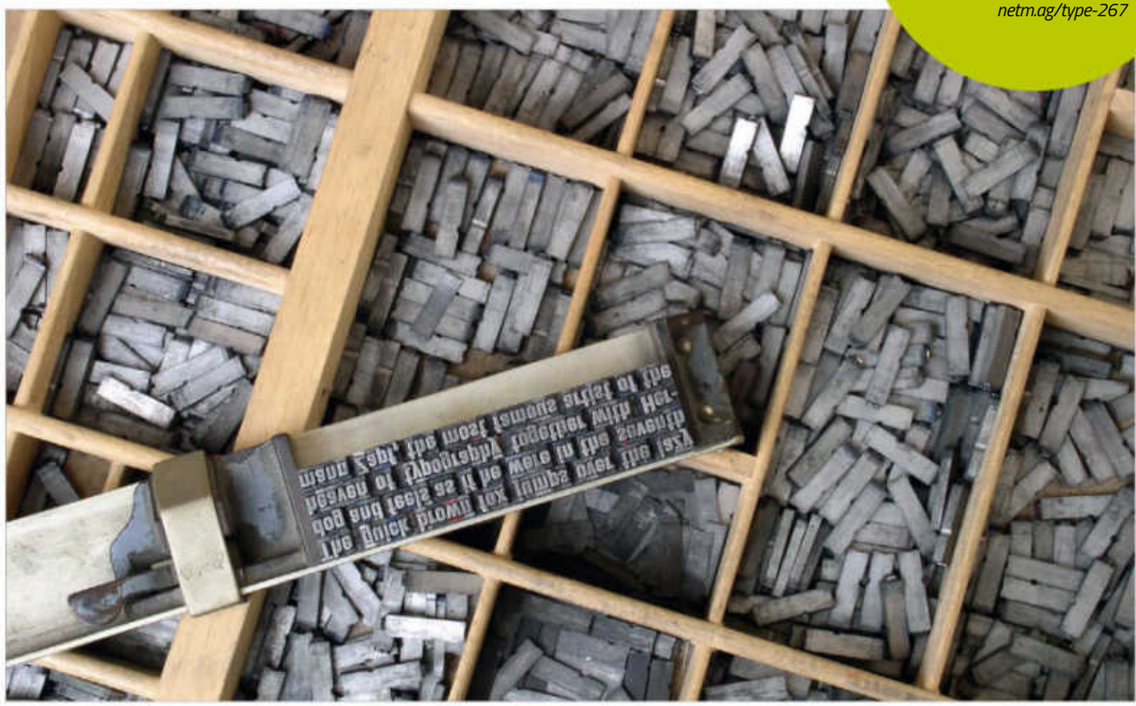


Image: Willi Heidelbach

TYPOGRAPHY

MAKE YOUR TYPE LOOK GOOD ON ANY DEVICE

Clarissa Peterson shares her tips for ensuring your text is easy to read, no matter what screen it's being viewed on

Typography can often seem like the easiest part of designing a web page – after all, how hard is it to put words on a page? But on responsive websites you are challenged with using media queries to make sure the type looks good and is easy to read, no matter the size of the user's screen. Your typography choices should help the website get its message across, not get in the way of the message. In this tutorial, I'll explain how to ensure your type works effectively on devices of all shapes and sizes.

SCALE

At any given screen size, the elements on the page need to be displayed in an appropriate scale. For example, on a small mobile phone screen you don't have a lot of space to work with, so it would look strange if everything was very large. That would also force the user to scroll more than necessary.

On the other hand, when your website is being viewed on a large monitor and there's plenty of space on the screen, you can take advantage of that fact by making individual elements larger, and by increasing the white space between them to make sure each element is distinct. If things are too small, or there isn't enough white space, the page will just look squashed and confusing.

Size, length and height

Font size works together with line height (leading) and line length (measure) to determine how the text fits in with the scale of the site. These qualities also affect the readability of the text. On responsive

BOOK

Learn more by checking out the chapter on responsive typography in Clarissa Peterson's book, *Learning Responsive Web Design: A Beginner's Guide*: *learningrwd.com*

White space On the ZURB blog, the amount of white space around the elements increases on wider viewports

websites, you should adjust each of these aspects using media queries to make sure your text always looks its best.

FONT SIZE

Many designers forget to consider the most important thing about font size: the text needs to be large enough for most users to easily read. Around age 40, people start to develop presbyopia, a condition which makes it hard to focus on things that are close. That means there are a lot of people (including me) that have difficulty reading small text on websites.

Font size works with line height and line length to determine how your text fits

You don't have to worry about what font size is large enough, because the browser makers have already figured it out for you. You just need to set your base font size to `100%`, and each browser will make that a font size most users will be able to easily read on that particular browser and device.

For most current devices (but not all), that `100%` font size is the equivalent of 16 pixels. Future types of devices are likely to have different base font sizes. By using `100%` now, you're getting ready for all those future devices.

Ems and rems

Set the base font size to `100%` in the `body` element. After that, all the fonts on your site need to be sized in ems or rems, which are relational units, rather than fixed units like pixels. This will make it easier to adjust the font size for different screen widths. For example, you can start out with your paragraph text at 1em, which is equal to the base font size. Setting your `H1` at 2em would make it twice as big as the base font size.

FOCUS ON

PERFORMANCE OF TYPOGRAPHY

The way you use fonts on your website can affect its performance. It usually doesn't have a big impact, so you'll probably want to worry about the big performance hits like images first. But if you want to squeeze a little more speed out of your site, taking a look at your fonts can help.

First, keep in mind that each typeface you use on your site requires the download of a file. The more typefaces you use, the more files there are to download. Many sites will look fine with only two or three different typefaces. If you need to differentiate between various types of content on the page, you can do so by adjusting the colour, weight and size of the text.

You can cut down on load time even more by using system fonts, which won't need to be downloaded. You can even use a media query to ensure system fonts are only used on narrow screens, which are more likely to be on a slow connection (although bear in mind this is not always the case).

```
p { font-family: Helvetica, Arial, 'Droid Sans', sans-serif; }
@media only screen and (min-width: 30em) {
        p { font-family: ShinyHappyFont, Helvetica, Arial, sans-serif; }
}
```

After using `@font-face` to declare the font you want to add to your site, the browser won't actually download the file until it's needed on the screen. So if the font is only called in a media query that doesn't apply to the current screen size, the browser won't download the font file. Not all devices have the same system fonts installed, so you need to make sure to include options for each mobile platform in your font stack.

Font	iOS 7	Android 4.4	WP 8	BlackBerry
Academy Engraved LET	Yes			
Al Nile	Yes			
American Typewriter	Yes			
Andale Mono				Yes
Apple Color Emoji	Yes			
Apple SD Gothic Neo	Yes			
AppleGothic Regular	Yes			
Arial	Yes		Yes	Yes
Arial Black			Yes	
Arial Hebrew	Yes			
Arial Rounded MT Bold	Yes			

System fonts Use Tinytype (*netm.ag/tiny-267*) to see which system fonts are available on each mobile platform

IN-DEPTH

ALIGNMENT AND HYPHENATION

Using alignment and hyphenation properly can make your text look better, especially on a small screen. Justified text (when your text goes all the way to the right margin on each line) will result in empty spaces in the middle of lines. Changing to left alignment gets rid of the spaces, but leaves a ragged right margin. Hyphenation can fix that.

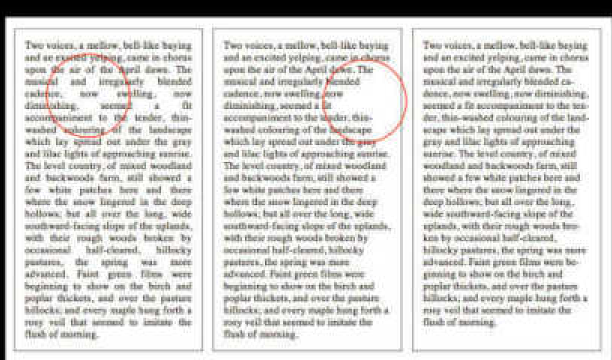
Hyphenation isn't supported in all browsers (not Chrome, Opera or Android), but that's OK. Hyphenation is an example of progressive enhancement. It's a nice add-on for the browsers that can do it, but users with other browsers won't be missing out on anything if they don't get it.

```
.hyphenate {
        -webkit-hyphens: auto;
        -moz-hyphens: auto;
        -ms-hyphens: auto;
        hyphens: auto;
}
```

In order for hyphenation to work, the browser needs to know what language the text is in. You can use the `lang` attribute on any element, but generally it's easiest to put it in the `html` element if the page is all in one language.

```
<html lang="en">
```

If you don't have a `lang` attribute, the browser will not hyphenate anything. Keep in mind that each browser can only hyphenate certain languages. See *netm.ag/hyphens-267* for more information on browser and language compatibility.



Closing the gap The justified paragraph (left) shows large gaps between words, while the non-hyphenated paragraph (middle) has a rough right margin

```
body { font-size: 100%; }
p { font-size: 1em; }
h2 { font-size: 2em; }
```

Until recently, a lot of websites used 10 pixels or 12 pixels as the base size for their paragraph text. If this is what you're used to, `100%` text might seem really large. But bigger text on a website shouldn't look wrong. If you look at the text in your design and it feels too big, that likely just means you need to adjust the scale of everything else to match the size of the text. Larger text means other elements on the page need to be larger, and there needs to be more white space between elements.

LINE HEIGHT

Line height is the distance between lines of text. It affects readability: if lines are too close together or too far apart, it will make it more difficult for your eyes to travel from the end of one line to the beginning of the next, and you'll have to put more effort into reading. A line height of 1 means the line is as tall as the text, with no extra space between lines. A line height of 2 means the empty space between lines of text is exactly as tall as the text.

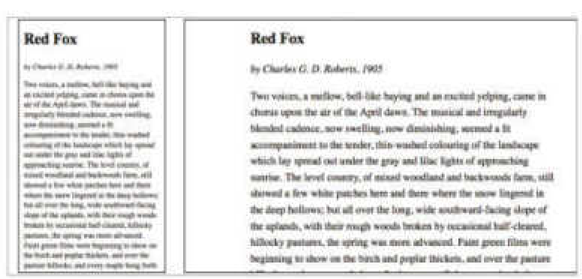
On average, the best line height for reading is around 1.4, so this is a good place to start. Keep in mind this applies only to blocks of text such as paragraphs, not to other text elements like headings. On small screens, your line height should be slightly less, and on large screens, slightly more.

Vertical margins, above and below paragraphs, should be in proportion to line height. You can do this by using the same number, but with a unit of em.

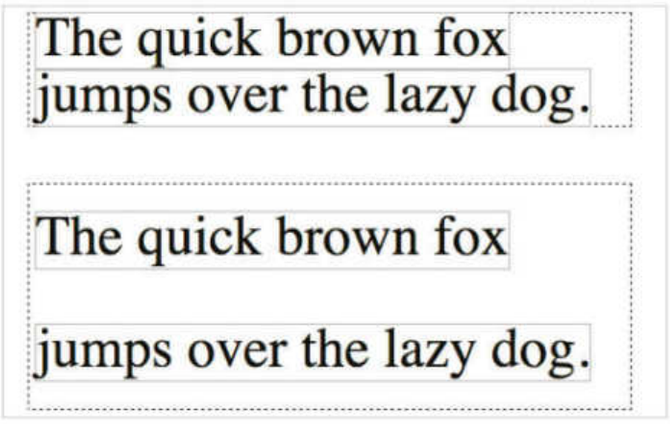
```
p { line-height: 1.4; margin: 1.4em auto; }
```

LINE LENGTH

Line length is very important to the readability of text. Very long lines are difficult to read, as your eyes have trouble moving from one line to the next. Short lines have a different problem: because your eyes are moving back and forth so often, you will tire of reading more quickly. Text is easiest to read



Tweaks and changes By adjusting your font size, margins and the number of columns, you can make text easy to read at any screen size



Left Here, the first paragraph has a line height of 1, and the second paragraph has a line height of 2

Right Use this bookmarklet from Chris Coyier (*netm.ag/coyier-267*) to see where the optimal range of 45-75 characters per line falls

if lines are on average 45–75 characters in length. Since each line will have a different number of characters, you're just shooting for an average. You can easily find out the if your text falls within that range by using Chris Coyier's handy bookmarklet (*netm.ag/coyier-267*). While working on a site, you may find it easier to simply add a `span` around the appropriate characters in your first paragraph and add a `class` to colour the text red. Starting with your design at the narrowest screen size, you'll likely have your text in a single column. At a font size of `100%`, your lines will be around 45 characters long, or slightly less. Looking at your design, slowly increase the width of your browser window and the line length will increase until it eventually hits 75 characters. This is the point at which you want to add a media query, to get your line length back within the 45–75 character range.

It's best to make smaller, incremental changes to typography at several breakpoints

Adjusting margins

There are a few things you can do to decrease your line length. First, you can increase your left and right margins. If you're working on a one-column website, this is the easiest option. Second, you can increase the number of columns, so your paragraph text doesn't go all the way across the screen. Third, you can increase the font size of your paragraphs, which will mean fewer characters per line.

You can use any combination of these approaches to make sure the line length stays within the 45–75 character range, no matter the width of the screen.

In the example below, I'm using arbitrary viewport widths – you'll want to pick the viewport widths that are appropriate for your design.

```
article p { font-size: 1em; line-height: 1.3; width: 96%; }
@media only screen and (min-width: 30em) {
        article p { width: 80%; }
}
@media only screen and (min-width: 50em) {
        article p { font-size: 1.1em; line-height: 1.4em; }
}
@media only screen and (min-width: 60em) {
        section { width: 70%; float: left; }
        article p { width: 100%; }
}
@media only screen and (min-width: 80em) {
        article p { font-size: 1.2em; line-height: 1.5em; }
}
@media only screen and (min-width: 100em) {
        section { width: 50%; }
}
```

Keep in mind you can have as many media queries as you want on a website. It's best to make smaller, incremental changes to typography at several breakpoints, rather than making huge changes all at once, at a couple of major breakpoints. Adjust the size of your browser window to see where media queries need to go – where the text starts to not look good, that's where you need a media query.

PUTTING IT TOGETHER

Keep in mind these aren't hard-and-fast rules. One key to making design work well is to test your site on a variety of devices at many different screen widths. Think about whether the font size, white space and line length look appropriate at each screen width. Making sure your text is easy to read will translate to more users reading more of your content, and more users returning to your site.

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/iframeGit-262*

ABOUT THE AUTHOR

JIMMY JACOBSON

w: *wedgies.com*

t: @jimmyjacobson

job: Co-founder and CTO, Wedgies

areas of expertise: JavaScript, NodeJS, Full-stack development, scalability

q: how do you deal with stressful situations?
a: Take a walk! When a situation or event is getting stressful, I like to take a quick walk around the block

JAVASCRIPT

ADD 3RD PARTY CONTENT TO RESPONSIVE SITES

Jimmy Jacobson explains how to use JavaScript, iframe and RWD techniques to embed third party web apps in your responsive site

Ah, the iframe tag. Before tables and CSS, it was all a web designer (or, as it was back then, a webmaster) had to control the layout of a page. People did crazy things with iframes, including trying to mimic the sidebar of Windows apps. As HTML evolved to support tables and then CSS, the iframe fell into disuse and even disdain. The only acceptable use of the iframe was to display ads.

Good news – the iframe is back! Actually, it never really went away. You just haven't been noticing it any more. Services like Disqus, Vine, CodePen, Tumblr, Wedgies and SoundCloud all use iframes to enable social media networks and third party websites to embed fully featured web apps. These apps offer users the ability to interact with content such as videos, user authentication and even social features – so please don't call them 'widgets'.

- Disqus is a commenting platform that turns any page into a forum thread with comments
- Twitter uses iframes generated via JavaScript to enable users to embed Tweets inside any web page
- Tumblr uses iframes to display videos and other content, as well as to prevent malicious JavaScript from being run inside its dashboard
- CodePen makes use of iframes to show real time previews of code being written by its users, and to embed those previews on third party websites

RESOURCE

3rd Party JavaScript by Ben Vinegar and Anton Kovalyov covers writing JavaScript to be used by third party websites. Check out the sections on writing your own library and cross-domain message passing: *netm.ag/3rd-262*

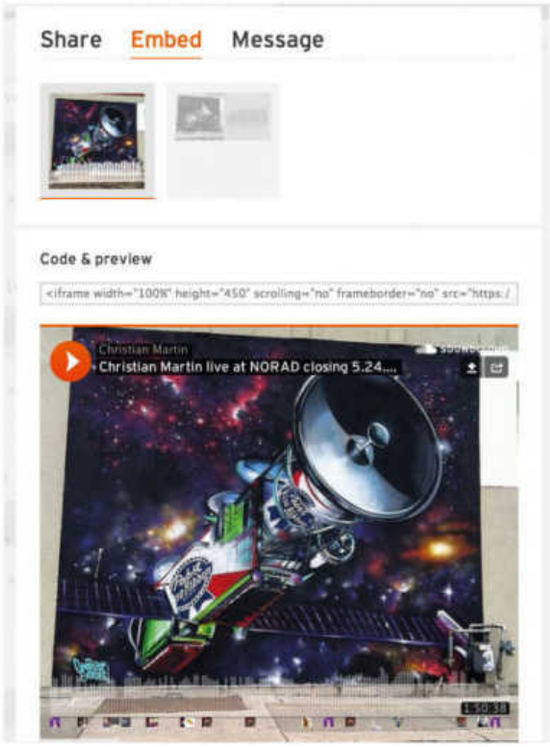
ADAPTING AND ADJUSTING

Living in the 21st century, content has to work on an array of devices. And that is true for these embedded web apps. Websites often use different layouts for mobile and desktop rendering. If the embedded web app does not adjust its display for different devices, it will not fit inside the mobile layout, ultimately causing the website owner or social network to abandon it.

These apps include the ability to interact with content – please don't call them 'widgets'

The solution to this is the same one that web designers and developers use for websites on mobile devices. Responsive design techniques, including using a fluid grid, flexible images and CSS media queries, work just as well when an HTML document is rendered inside an iframe as they do when the document is the root of the window.

There is one glaring issue that keeps this from being a perfect solution. It is impossible for an iframe to have a height specified by a percentage,



Set height The UI for embedding a SoundCloud shows you can't set the height of an iframe to 100 per cent via its attributes

and for an iframe to detect the height of the content being rendered inside it. In most cases, when an explicit height in pixels can't be detected for an iframe, browsers default to 150 pixels.

FINDING A FIX

At Wedgies, our solution to this problem comes from the excellent book *Third Party JavaScript* by Ben Vinegar and Anton Kovalyov (*netm.ag/thirdparty-262*). Using JavaScript, an HTML document embedded in an iframe can pass a message to the parent window using cross-domain messaging.

The flow is pretty simple, but there are some specific things that have to be done in order to conform to the security policies of various browsers. Internet Explorer, in particular, has some very strange requirements (see the GitHub repo at *netm.ag/iframeGit-262* for more on this).

1. Register an event handler in the embedded document for resize events
2. The embedded document posts a message to the parent window that contains its own height. This height is detected via the height attribute of the embedded document in the resize event handler
3. The parent window listens for the message and resizes the iframe element whenever the message is detected, using the data in the message

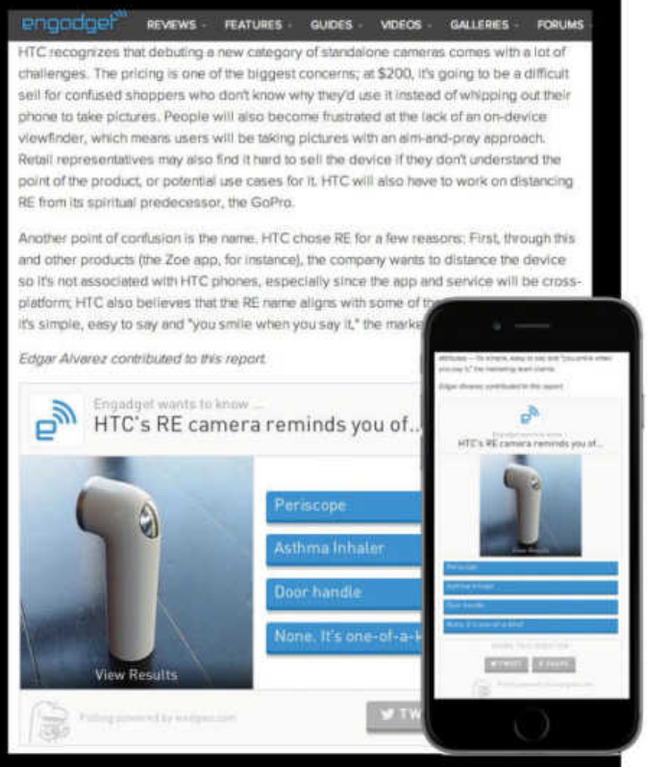
CASE STUDY

EMBEDDING POLLS

Wedgies (*wedgies.com*) provides a platform for creating and voting on polls. A lot of Wedgies' users choose to embed these polls on their own websites. This is how the development team at Wedgies learned how difficult it could be for a user to embed third party content into a responsive layout.

Wedgies experimented with a few different ways to make this process easier on the user, including allowing the user themselves to set a height and width on the content. But in the end we found the best solution, as ever, was the one that removed the worry from the end user. Just make it work.

The technique we now use is achieved through a combination of JavaScript that is loaded into the user's website, and responsive design in the content provided to the iframe. Below you can see two examples of one poll embedded into both a desktop layout and a mobile layout. The content has adapted responsively and the iframe has been resized automatically. In both examples, the code provided to the end user and the code served into the iframe are identical.



Changing shape Shown here is a Wedgies poll embedded into a page rendered on a large screen. Next to it is the same poll embedded into the same page, but rendered in a mobile screen size

Take a look at the following code, which goes in the webpage that is embedded via iframe. Use of jQuery is assumed in this example.

```
function postMessage() {
var msg = $('body').height();
window.parent.postMessage(msg, '*');
}
```

Event handler for window resize:

```
$(window).resize(postMessage);
```

Event handler for page load:

```
$().ready(postMessage);
```

This code is loaded by the parent webpage. It listens for a message and sets the height of the element with `id = id-of-iframe` to the value in the message. This is pretty simple code.

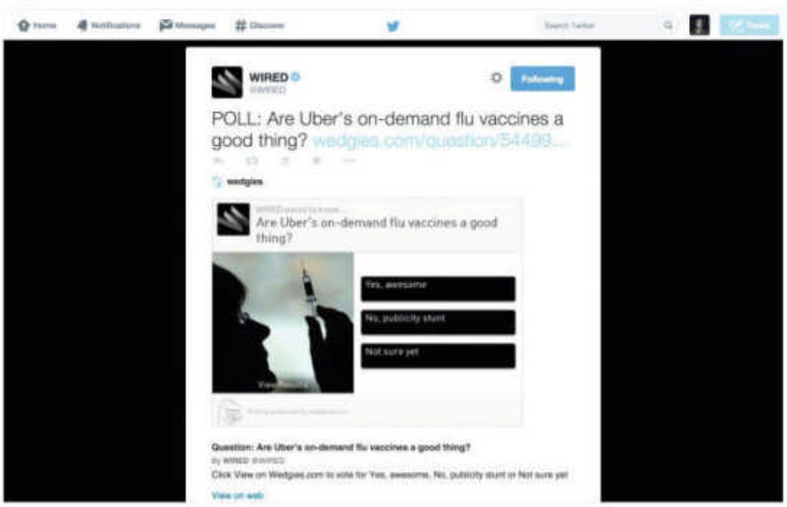
In a production environment there are many things posting messages, so be careful about filtering for the specific message from your app.

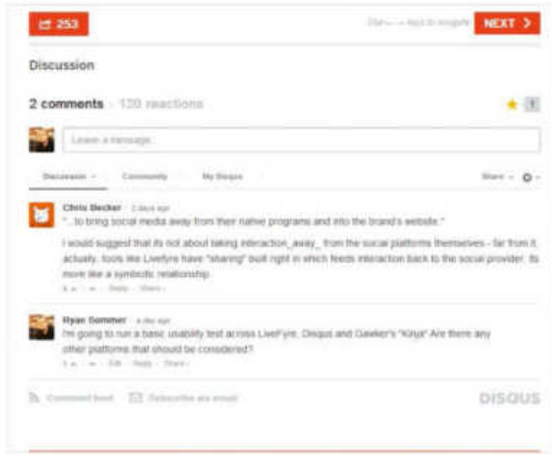
```
function msgReceiver(ev){

var msg = ev.data;

var elem = document.getElementById('id-of-iframe');
if (elem) {
elem.style.height = (parseInt(msg)) + "px";
}
}
```

This technique requires the developer of the embedded web app to have control over JavaScript on the parent page. The best way to do this is to give





Top Disqus is deployed to third party websites via JS, and generates an iframe to load the web app

Below Wedgies uses PhantomJS to render the embedded web app view of a poll, which can be used as content inside Facebook and Twitter posts

them a combination of a JS library and a snippet of HTML containing a unique ID. When run, the JS library detects these snippets and expands them into the code needed to render the iframe containing the correct content. The library also contains the message listener and code to resize the iframe when a message is received.

This technique allows the iframe to be resized along with the page, and for the document embedded in the iframe to respond to the resize events responsively. Now when a user pulls up the web page on a mobile device, the embedded web app responds to the width and height correctly, instead of being rendered in a fixed height iframe, cutting off the content in the mobile layout.

BENEFITS AND BONUSES

Leveraging responsive design inside a web app intended to be embedded on third party websites has other bonuses as well:

- By providing an oEmbed endpoint, the web app can be discovered by services like Embedly (*embed.ly*) that aggregate embeddable services, exposing the app to more potential users
- Building plugins on platforms like WordPress is much easier
- PhantomJS can be used to render a screenshot of the embedded web app and used as content for Twitter or Facebook posts
- Content is ready to be used inside a mobile app via Web View, without any modifications

RWD techniques aren't just for mobile. iframes trigger the same events and constraints a document needs to render itself responsively. A web app deployed to an iframe and distributed across websites will have broader adoption if it works across any screen size or mobile device.

ACCESSIBILITY

ACCESSIBLE RWD

Derek Featherstone gives his four top tips for ensuring your responsive sites and apps are as accessible as possible

Responsive design and accessibility both embrace the idea of flexibility. But there's more to it than that. Simply implementing responsive design techniques doesn't automatically make your site accessible. Here are four practical tips to help boost accessibility in your responsive designs.

TEST WITH REAL DEVICES

Responsive testing tools, bookmarklets, extensions and testing platforms do a good job of helping test different sizes. They do nothing to help you learn about how your designs work with assistive technology. You need to test with real devices, and preferably with real people.

BEWARE LOST FOCUS SYNDROME

On a mobile or tablet where there is no physical cursor, the screen reader cursor 'floats' above the screen. Try this: create a note, turn on VoiceOver and flick through so you can see the cursor on some text. Now, rotate your device. Notice the cursor doesn't stay on the object that had focus. Instead, it gets 'lost'. It may find a nearby object, or jump to the top-left of the screen. This often happens when showing and hiding content, or when you animate a menu and the content slides past the VoiceOver cursor.

Often the cause is content that is off-screen but is focused. Be sure to test with real devices to activate menus and other items. If you find that you're losing focus, hide off-screen content fully with `display:none` rather than just placing it off-screen.

MIND THE OVERLAP

This isn't unique to responsive designs, but it certainly plagues them. Similar to Lost Focus Syndrome, we often see the VoiceOver or TalkBack cursor on a link or other object, but that link is partially obscured or overlapped by some other content. When we activate the link with a double-tap, we end up activating the other piece of content (see *sateach.es/mind-the-overlap*). Test your work and look for areas of overlap. If you find activating an item actually activates a different item to the one you intended, modify your CSS so that the items don't overlap, or are hidden with `display:none`.

CONSIDER VIEWPORT SIZE

With responsive design, we're designing for different viewports and making assumptions about the device based on size. Remember, just because the viewport is 800x600 pixels doesn't mean the user is on a tablet. I've seen people with low vision using a desktop 27" screen with 800x600 resolution so they can see it. Take care not to make assumptions about device capabilities or how that person will interact with the site simply based on viewport size.

PROFILE

Derek (@feather) is an internationally recognised speaker, entrepreneur and founder of Simply Accessible (*simplyaccessible.com*). He is also a respected authority on accessibility and interaction design

ABOUT THE AUTHOR

GION KUNZ

w: *mindtheshift.wordpress.com*

t: @GionKunz

areas of expertise:
Frontend development

q: who would play you in a movie of your life?
a: The guy who played Steve Urkel!

CHARTS

RESPONSIVE CHARTS WITH CHARTIST.JS

Gion Kunz shows you how to use Chartist.js to create great-looking, responsive charts in your next data visualisation project

As a responsive web developer I'm constantly seeking ways to bring standard technologies under one hood, while also trying to keep up with the W3C One Web promise (*netm.ag/oneweb-261*). Making the same information, design and interactions available on every medium – despite each coming with their own particular restrictions – may seem impossible at first. It's one of the most challenging parts of a developer's job, but I still think that this is a promise we can and should fulfil.

The problem with not displaying some content on certain media, or dismissing some interactions, is not really that it breaks a W3C contract, but that it messes with our users' expectations. That's where we pay the fine. There's nothing more annoying than discovering your favourite button on a website does not exist when you're sitting in the bus and feel like pressing it! In my experience this can even cause a user to simply choose not to view a website on a given media completely.

I think these principles should be applied to any content on the web, and this includes data visualisation. Faced with a project for a client who wanted me to implement a performance dashboard that should be accessible on a mobile and tablet, but also displayed nicely on a big widescreen television, I felt lost with the options available.

There were already tons of good charting libraries out there, but none of them actually solved the issue of generating simple and nice looking charts that also behaved responsively. After spending hours trying to tweak existing libraries to make them behave how I wanted them to, I simply decided to create my own. This marked the birth of Chartist.js (*netm.ag/chartist-261*).

GETTING UP AND RUNNING

The easiest way to started with Chartist.js is by using Bower package manager (*bower.io*). You can then install the latest version of Chartist.js by typing the following command in your shell:

```
bower install chartist --save
```

You will now have Chartist.js installed locally, and awaiting your further instruction. All you need to do is add the resources into your HTML and you're ready to draw some nice responsive charts.

```
<link rel="stylesheet" href="bower_components/chartist/libdist/chartist.min.css">
<script src="bower_components/chartist/libdist/chartist.min.js"></script>
```

Chartist.js comes without any dependencies and has a compressed size of less than 10KB. The core purpose of Chartist.js is to solve one and only one problem, which is to enable developers to draw simple, responsive charts. It does that using the power of web standards, like inline SVG in the DOM, and CSS for its appearance. With this clear separation of concerns, and leveraging standard technologies, Chartist.js is able to provide you with all you need and nothing more.

The biggest issue I'm currently noticing with a lot of libraries is that they are trying to solve too many

BROWSER SUPPORT CHARTIST.JS

Desktop	Mobile/tablet
9	5.1
6	3
9*	-
12.1	37
5.1	32

* With matchMedia Polyfill

problems, and forgetting about the fact that they are libraries and not applications.

I'm sure you've run into situations where a library customises everything but the one thing you'd really like to customise. By building upon web standards, you can easily extend the functionality of Chartist.js and scale linearly.

HACKING YOUR FIRST CHART

Now as you've included Chartist.js into your project, you can go ahead and create your first chart. The library comes with some default styles that can easily be overridden or customised with the Sass (SCSS) version.

To get started, you can simply add a new element within your HTML and give it an ID. This will create a chart container.

```
<div id="my-chart"></div>
```

That's already it for the HTML part. Of course it's up to you if you'd like to use a `<div>`, or any other element you think matches the context better.

Chartist.js tackles only one problem: to enable devs to draw simple, responsive charts

Now let's go ahead and initialise a simple line chart with two series by using the function `Line` in the global Chartist.js namespace.

```
Chartist.Line('#my-chart', {
  labels: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri'],
  series: [[0, 3, 2, 8, 9], [1, 2, 3, 5, 8]]
}, {
  width: '300px',
  height: '200px'
});
```

Congratulations, you've just created your first chart using Chartist.js! Wasn't that a piece of cake? But wait … isn't that supposed to be a responsive charting library? Of course I wouldn't recommend that you use fixed dimensions for your chart, based on pixels, right?

ASPECT RATIOS AND SCALES

Because of the nature of responsive design, it's important to understand that block content in design, including images, videos and the like need to be able to scale and adapt to the medium. In order for an element to scale, you need to rely on certain aspect ratios (4:3, 3:2, 16:9 and so on) rather than specifying a fixed width and height.

With Chartist.js you can specify those ratios directly on containers, without the need to calculate any fixed dimensions. In order to create a chart that uses the aspect ratio of a golden section, you can just add the class `.ct-golden-section` to your container where you initialise your chart.

There are also classes for other common scales that you can use instead. Just check the Chartist.js documentation online (*netm.ag/chartistdoc-261*) to get a complete list of aspect ratio classes you can use, out of the box.

```
<div class="ct-chart ct-golden-section"></div>
```

You can then just omit the fixed width and height in the options when initialising your chart.

```
Chartist.Line('#my-chart', {
  labels: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri'],
  series: [[0, 3, 2, 8, 9], [1, 2, 3, 5, 8]]
});
```

Your chart will now use 100 per cent of the available width and scale with the golden section as fixed aspect ratio.

RESPONSIVE SUGAR TOPPING

Chartist.js provides an easy way to specify configuration that should be used specifically for a given medium. It makes use of CSS media queries and `window.matchMedia` to provide a configuration override mechanism that allows you to fully re-configure your charts based on media query rules.

The following example shows a mobile-first configuration style where small screen devices will see a point on the line chart and on larger screens the points will not be drawn.

```
Chartist.Line('#my-chart', {
  labels: ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday'],
  series: [[12, 9, 7, 8, 5]]
}, {
  showPoint: true
}, [['screen and (min-width: 640px)', { showPoint: false }]]);
```

I hope this has given you a good overview of the philosophy of Chartist.js, and that you're hopefully already thinking about a next project where you could use it.

RESOURCES

LIVE CODING

Visit the Chartist.js website to check the latest updates, and hack on some charts in the browser using the live example page. *netm.ag/livecoding-261*

ABOUT THE AUTHOR

SHANE OSBOURNE

w: *wearejh.com*

t: @shaneOsbourne

job: Web developer, JH

areas of expertise: JavaScript, Node.js, frontend tooling

q: what's the strangest thing you've ever eaten?
a: Although I'm a veggie now, I once ate a caramelised Scorpion whilst visiting my father in Thailand

TESTING

TEST YOUR SITES ON MULTIPLE DEVICES

Shane Osbourne explains how Browsersync helps you test your sites easily and effectively on multiple devices and browsers

There's no denying that, for a growing percentage of users, the primary means of internet access is some form of mobile device – be it a smartphone, tablet, notebook or anything in-between. As designers and developers, you'll already be aware that these devices may or may not be touch enabled, have varying screen sizes and pixel densities, offer an unknown (at least until page load) set of supported features and may be connected to any type of network ranging from an extremely slow EDGE connection to a super-fast Wi-Fi connection.

You should also be well aware that testing your website on as many of these actual devices as possible is the only way to ensure your designs and code work smoothly and correctly for as many people as possible.

Browsersync (*browsersync.io*) is a free, open source project, which runs on Windows, Linux and Mac OSX. It was designed from the ground up with exactly this type of multi-browser, multi-device testing workflow in mind. As such, it makes the whole process a breeze.

Browsersync has many amazing features, but there are three fundamental ones that directly address the most common pain-points for testing:

1. Accessing a localhost website on all of your test devices whilst in development
2. Auto-reloading all devices when you change files
3. Synchronising interactions across all devices

I'll take a look at each of these key features in turn in this tutorial.

SETTING UP

First up, install Browsersync as per the guidelines on *browsersync.io*. You'll also need to set up a new folder, create an 'index.html' file and a CSS folder containing the file 'style.css'. The contents of 'index.html' would look something like this:

```
<!doctype html>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>Browsersync example</title>
  <link rel="stylesheet" href="css/style.css"/>
</head>
<body>
  <h1>Browsersync example</h1>
</body>
</html>
```

PRO TIP

As that almost anything you can imagine can be built entirely with JavaScript these days (thanks to the browser and Node.js) I'd recommend it as a top language to learn!

ACCESSING A LOCALHOST WEBSITE

Browsersync has the ability to create a mini server from any folder on your computer and expose that server over your Wi-Fi network. On the command line, navigate into the folder that contains the 'index.html' file and run the following:

```
$ browser-sync start --server
```

You'll be presented with two URLs: a 'localhost' address and an IP-based address. The localhost address is the one you typically use on the same machine that you develop from, while the IP-based address is for any device that's connected on the same Wi-Fi network.

Using these two addresses, you now have access to your website on any phones or tablets you can get your hands on (as well as your main development machine). This means you can spot and fix issues with your layout, CSS and JavaScript as early as possible in the development process.

Browsersync was built with a multi-browser, multi-device testing workflow in mind

If you're in a situation where you already have a server setup (such as WordPress running on MAMP/WAMP, for example), then you would replace the `--server` option for `--proxy` and provide your existing URL:

```
$ browser-sync start --proxy http://local.wordpress.com
```

AUTO-RELOADING DEVICES

Now you can easily access your work-in-progress website on any device connected to the same Wi-Fi network. However, you'll quickly realise that this type of workflow actually introduces a new issue: viewing the latest version each time you change a file requires you to reload all browsers on all devices manually. This is where Browsersync's built-in File Watcher comes in handy. Not only does it reload all browsers automatically when it notices a file change, but it can actually inject CSS changes directly, without even reloading the page.

Building upon the previous server example, you can provide a `--files` flag with glob patterns that match your project files. By doing this, you are instructing Browsersync to watch a subset of files for changes and, when it sees a change occur, either perform a hard reload or inject changes into all connected browsers automatically.

```
$ browser-sync start --server --files="*.html, css/*.css"
```

With the above command, any changes to 'index.html' will cause every browser to perform a hard reload, whereas changes to 'css/style.css' will be injected into the page *without* reloading it. Now you can go ahead and start coding your latest creation – each time you hit Save, you'll be able to instantly see the changes on each and every device simultaneously.

Once you've experienced the level of efficiency this workflow affords – with its instant feedback and shorter iteration times – you'll find it almost impossible to believe you ever managed to work without it!

SYNCHRONISING INTERACTIONS

The final piece of the awesome-workflow puzzle is the synchronisation of actions across all connected devices. We realised early in the development of Browsersync that being able to scroll inside one browser and see that action mirrored in all others is not simply a cool party trick, but actually an essential counterpart to the auto-reloading feature. The Action Sync feature (enabled by default) will copy scroll position, clicks and form inputs from one browser to all of the others that are connected.

A perfect example of why this is useful would be a mobile slide-out navigation. You could have your site open on a multitude of devices that all fall below a breakpoint that causes the navigation to be hidden. Now, when you tap or click the 'menu reveal' button on any of the devices, that same click will be replayed across them all – thus revealing your navigation and highlighting any layout or CSS bugs that may be different across platforms.

OTHER HELPFUL FEATURES

The three features we've just looked at form the basis of Browsersync, but that's just the beginning. By accessing the browser-based UI that comes bundled with Browsersync, you'll find a range of additional features. These include 'remote debugging' (like Chrome's 'inspect element', but for all browsers), 'URL history' (records your test URLs so you can push them back out to all devices with a single click), 'sync customisation' (toggle individual sync settings to create your preferred test environment) and 'URL tunnelling' (create a secure public URL to share your local sites with any internet-connected device). So get stuck in!

View source files here!

All the files you need for this tutorial can be found at *netm.ag/emailgit-269*

ABOUT THE AUTHOR

JENNIFER WONG

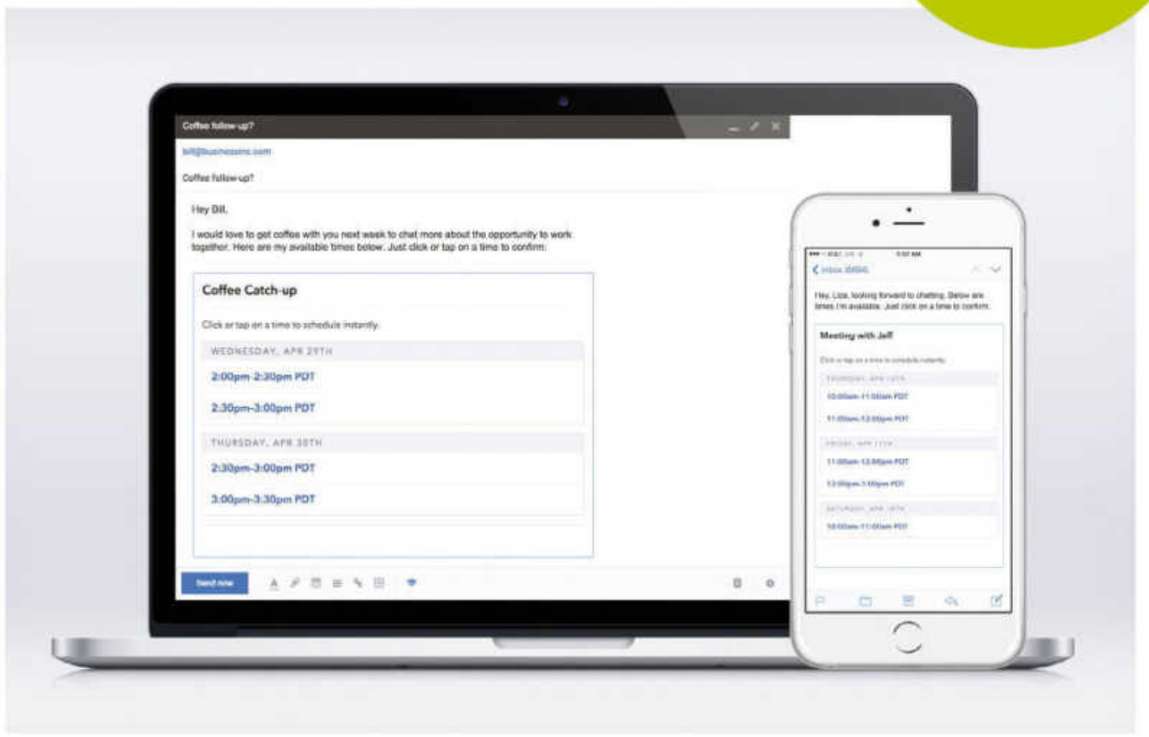
w: *mochimachine.org*

t: @mybluewristband

job: Frontend developer

areas of expertise: HTML, CSS, Sass, JavaScript, jQuery, RWD, responsive email design

q: what's the strangest thing you've ever eaten?
a: Geoduck sashimi



EMAIL

MASTER THE TRICKS OF RESPONSIVE EMAIL

Jennifer Wong tackles tables, inline styles and inconsistent CSS support to show you how to design emails fit for 2016

It's the moment you've been dreading: the project of redesigning all consumer-facing emails and making them responsive becomes yours. And you've heard the rumours – designing emails means coding like it's 1999, creating tables and adding styles inline (heaven forbid), and throwing best practices and hopes of compatibility out of the window. Unfortunately, the rumours are true.

For starters, tables are the only reliable way to create consistent email layouts. Inline styles are a requirement because many email clients do not support external style sheets, and Gmail doesn't support the `<style>` element. Add to that inconsistent CSS support across email clients, and it seems like responsive emails are the makings of disaster.

Not to worry! In this tutorial, I'll help you get your emails in shape for 2016 and ready for the responsive spotlight. I'll show you:

- What's so important about design – and how a bit of research will save you a ton of time
- Why you need a reusable, maintainable template (or three), and how to design one
- When and how to use media queries and the ever-controversial `!important`
- How to make desktop, mobile and web-based clients play nicely together (and which ones to watch out for)

VIDEO

Jennifer Wong has created an exclusive screencast to go with this tutorial. Watch along at *netm.ag/emailvid-269*

So let's get started. When starting the design of an email, the first, most important step is to establish your audience. Where do they check their emails and what types of devices do they use? How much time do they spend viewing each email? Are they more visual or analytical?

You need to determine the devices and clients on which your users view emails. Targeting specific devices and email clients confines the actual work of implementation, saving you time. For example, knowing the majority of your users view their emails on mobile devices means you can spend more time focusing on mobile email design and less on desktop. Or if you know that only 0.5 per cent of customers use Outlook 2000, you might decide not to support that email client at all.

DESIGN ELEMENTS

Knowing which devices most of your customers use helps you design for a specific minimum touchable area (MTA). This can be thought of as the size of a button for mobile devices.

Knowing what devices most of your customers use helps you design for a specific MTA

You'll often hear 44px by 44px thrown around in mobile design discussions about MTA. Unfortunately, that number comes from the Human Interface Guidelines for the first iPhone. In other words, it's severely out of date. Accurate numbers should be based on the average size of a human's thumb (about 0.4 inches, according to one study: *netm.ag/thumb-269*) and the pixels per inch (PPI) of the device.

A quick calculation will give you the minimum pixel dimension (MPD) for the size of your buttons:

```
0.4" x Device PPI = MPD
```

PugetWorks provides some helpful numbers here: *netm.ag/puget-269*.

Content above the fold is another crucial aspect of responsive email design to consider. Your most important information should be conveyed in the initial viewport of any device. If your hero image is too tall and has huge margins, your title text may be pushed below the fold on a mobile device.

It's even possible that people won't think to scroll down, thereby disregarding your email entirely. Keep this in mind when designing your emails and be sure ▶

IN-DEPTH

OUTLOOK ISSUES

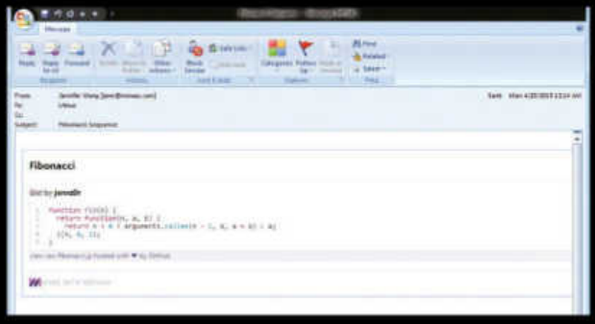
Something to keep in mind when developing responsive emails is the lack of CSS support in Outlook 2007, 2010 and 2013. The reason behind this is their rendering engines.

- Outlook 2000-2003: Internet Explorer
- Outlook 2011, 2015: WebKit
- Outlook 2007, 2010, 2013: Microsoft Word

Some key things that the Word-rendered Outlooks don't support are media queries, `position`, `display`, `height`, `max-width`, `min-width`, width on `<p>` and `<div>`, and padding on `<p>`, `<div>`, and `<a>`. However, take heart – there are some workarounds to help you.

For example, when trying to shrink a double-sized image for Retina devices, you're unable to use `width` or `max-width`. The workaround sets the original (non-high-resolution) `<img>` to `display: none;` and the high resolution image as a `background-image` with `height` and `width` on the surrounding `<td>` when viewed on Retina devices.

```
@media all and (min-device-pixel-ratio: 1.5) {
 td[class="pillows"] {
  background: url('http://mochimachine.org/images/pillows-hi-res.jpg') 30px 30px no-repeat;
  width: 30px !important;
  height: 30px !important;
 }
 td[class="pillows"] img {
  display: none;
 }
}
<td class="pillows">
 <img src="http://mochimachine.org/images/pillows.jpg" width="30px" height="30px">
</td>
```



CSS support Here you can see Outlook ignoring max-width, margin, padding and border style properties

to use an email testing suite or a library of devices so you can see what content is immediately viewable across devices and email clients.

Once you have an initial design, it's a smart idea to iterate. A/B tests determine what types of emails and designs are most effective for your users. It's a common belief that more images mean more engagement, but sometimes text-heavy emails result in better click-through rates. The only way to know is by testing the heck out of your emails.

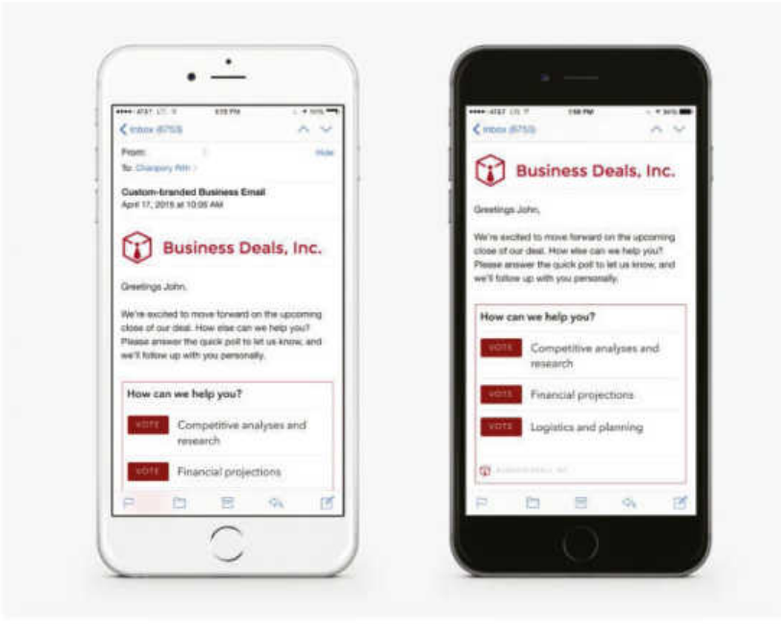
USING TEMPLATES

Templates save time by following the DRY principle. They also allow for abstraction of code and the reuse of styles and snippets over several types of emails. You'll likely have one overarching (or main) template that contains base styles for all of your emails, snippet templates for pieces of code that appear in most of your emails, and several unique templates that each represent one email.

Think of your main template as a sort of container for your emails. Use `max-width` of 550–600px, a number based on the typical preview pane of email clients. With `max-width`, content will stay enclosed within those bounds, ensuring that users won't have to scroll or scan from side to side. Note that `max-width` is not supported by Outlook 2007, 2010, or 2013 (see boxout on page 201).

Next, consider what design elements you're likely to reuse in most emails. For example, you'll probably have the same header, footer, and social media calls to action in many of your emails. These will be your snippet templates. For snippets, use a templating system to break those portions of code out into their own separate files. Then every email that uses those snippets can include an abstracted file in its unique template, saving you from typing the same code over and over again.

At a previous job, we used Handlebars.js and had a snippet (or what Handlebars calls a 'partial') for a badge that was repeated in our email widgets. So I had a snippet file called `_badge.hbs` and included it using the syntax for partials: `{{> _badge}}`.



Mind the fold Apple Mail sometimes displays email details such as the sender, subject and time, pushing content below the fold

TACKLING TABLES

Mozilla Developer Network states, "Prior to the creation of CSS, HTML `<table>` elements were often used as a method for page layout. This usage has been discouraged since HTML 4, and the `<table>` element should not be used for layout purposes" (*netm.ag/MDN-269*). But when it comes to email, we know better: tables are necessary for layout.

Without an outer `<table>`, the entire email will disappear from Apple Mail and Outlook 2011. But even if we do use an initial outer `<table>`, that's just the beginning. The more complicated the layout, the more `<table>`s you'll need.

Since emails contain a multitude of nested tables, comments often save the day. Add comments stating the purpose of a table at the beginning and end of the table's HTML to help you and your colleagues quickly read your email code.

```
<table cellpadding="0" cellspacing="0" valign="top"
style="border-collapse:collapse; width:100%; max-
width:600px;"><tr><td>
 <!--Inner Table Begin-->
 <table cellpadding="0" cellspacing="0" valign="top"
style="border-collapse:collapse;"><tr><td>
  <!--Content Begin-->
 </td></tr></table>
 <!--Inner Table End-->
</td></tr></table>
```

When designing my first template, I discovered that you need to set `<table>` to `display: block;` or email clients on Chrome will ignore your `max-width`. This is something to keep in mind if you notice problems with your table widths.

INLINE STYLES

As mentioned above, because of inconsistent `<style>`, `<link>`, and `<head>` support, inline styles are necessary to style your email. You can save time by finding a CSS inliner tool that will translate styles from a `<style>` or external style sheet, so you don't have to write CSS in multiple places.

There are many inliner tools to choose from on the web. Well-known companies like Campaign

Monitor, MailChimp and Zurb, as well as generous individuals in open source, provide CSS inliners for free. Most just require you to paste HTML with a `<style>` onto their site. For a more permanent solution, some tools can be incorporated directly into your project as a plugin.

Alternatively, always writing your styles inline typically allows for quicker prototyping, which will also save you time. When your styles are inline throughout your coding process, it's easy to change them right in the HTML and immediately see how the email rendering is affected.

When debugging email code, keep in mind inline styles are highest in CSS hierarchy, with the exception of styles that use `!important`. So when you're trying to find the origin point of styles, you'll have to check your external style sheet, `<style>` in HTML, and inline styles, then do a quick search for `!important` in each of those locations.

GOING RESPONSIVE

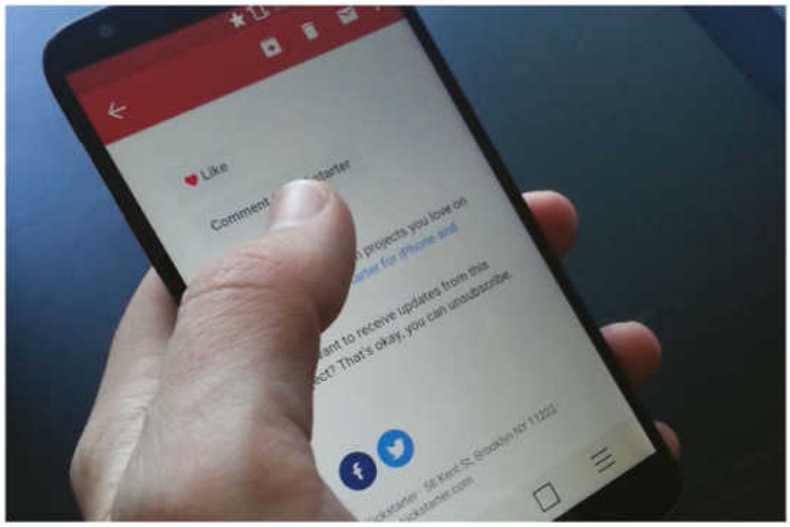
Responsive web design depends heavily on media queries. These target different media types, widths, device pixel ratios (for high-resolution) and more. While not completely useful with email, media queries are supported on iOS 7, iPad, AppleMail, and webmail clients.

Because of inconsistent <style>, <link> and <head> support, inline styles are necessary

There are several breakpoint widths associated with specific devices, which you can use to your advantage if you want specific content to be shown or hidden at different times (CSS Tricks has a great article on this: *netm.ag/breakpoint-269*). Again, don't forget to take note of which email clients do or do not support media queries.

When designing responsively, good practice means pairing media queries with fluid layouts. Fluid layouts use percentage-based widths so content flows to fit smaller screens. Often in email design, you'll see a `max-width` in `px` dimensions paired with a `width` of `100%`. This strategy confines desktop email clients to a `max-width` of 550–600px, but allows the email to fill the screens of mobile devices that are under 550–600px.

As mentioned previously, Outlook 2007, 2010 and 2013 ignore `max-width` and `min-width`. You'll therefore need a bypass in order for emails to display correctly. This bypass employs fluid layouts for most clients and a fixed (instead of max) width within a media query for widescreen desktop clients. This is another great example of pairing fluid and responsive layouts.



Thumb size A responsive email with great spacing, readable text, and button sizes that meet the minimum touchable area for this device

```
.background-table {
  width: 100% !important;
}
@media only screen and (min-device-width: 768px) and
(max-device-width: 1024px) {
  .background-table {
    width: 600px;
  }
}
```

Often developers say that `!important` should never be used. However, inline styles overwrite styles contained in media queries. In this case, the only way for the styles in a media query to take precedence over inline styles is by using `!important`.

```
@media only screen and (max-width: 480px), (max-device-
width: 480px) {
  .header {
    height: 20px !important;
  }
}
<table><tr><td class="header" style="height: 50px;">
  <h1>Cats Totally Rule</h1>
</td></tr></table>
```

Note that if you decide to use `!important` on inline styles, Outlook 2007 will completely ignore styles assigned that way.

RESOURCE

Jennifer Wong has created a customised wrapper <table> template for responsive emails. Check it out at *netm.ag/template-269*

FOCUS ON

RESOURCES

It's dangerous to go it alone. Tons of great resources like blogs, testing suites and templates are there to help you on your way. Reading email-focused blogs is a great way to keep up to date on the latest trends and workarounds. Testing is essential to the viability of responsive emails. If you lack experience designing emails, pre-made templates can be a great starting point. Here's what I recommend:

BLOGS

Style Campaign (*stylecampaign.com*) **–** This email creative agency run by Anna Yeaman produces amazing, practical content.

Campaign Monitor (*campaignmonitor.com*) – An email marketing campaign solution, this has over 10 years of content and maintains a thorough, up-to-date CSS-support webpage.

Litmus (*litmus.com*) – Litmus employs several content creators who constantly publish what's new in responsive email.

TESTING

Litmus – This has the largest desktop and mobile client coverage, including Lotus Notes and Blackberry, and includes access to its active Community forums.

Email on Acid (*emailonacid.com*) – This offers extensive web client coverage, even including sites like *Mail.ru*, and includes tips and tricks.

Targeted.io (*targeted.io*) – The newest and most minimal, but a huge plus is its affordable pricing.

TEMPLATES

Ink (*zurb.com/ink*) – Creators of responsive framework Foundation, Zurb has released these responsive email templates. Note that Ink doesn't support Outlook 2007, 2010 or 2013.

Campaign Monitor – As mentioned above, it also provides over 100 email templates for free.

MailChimp (*mailchimp.com*) – Another email marketing campaign solution, which provides free email 'blueprints'.

EMAIL QUIRKS

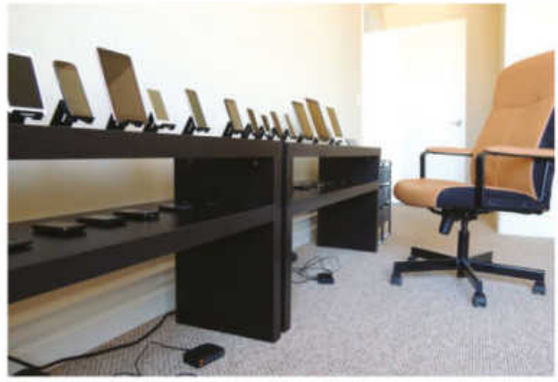
Email clients do a lot of weird things. Some of these you can prepare for. For example, many clients automatically hide images and require users to enable the display of them. To combat this, it's a great idea to include an `alt` attribute and/or `background-color` style for each of your images. That way, at least some content is temporarily displayed for the users in place of your images.

Other times, email client quirks are completely unpredictable. For instance, white space in your email code can sometimes affect rendering. I once noticed that images that were displayed in Gmail included a thin grey border. I could not for the life of me determine the cause. Eventually, I discovered that removing the white space

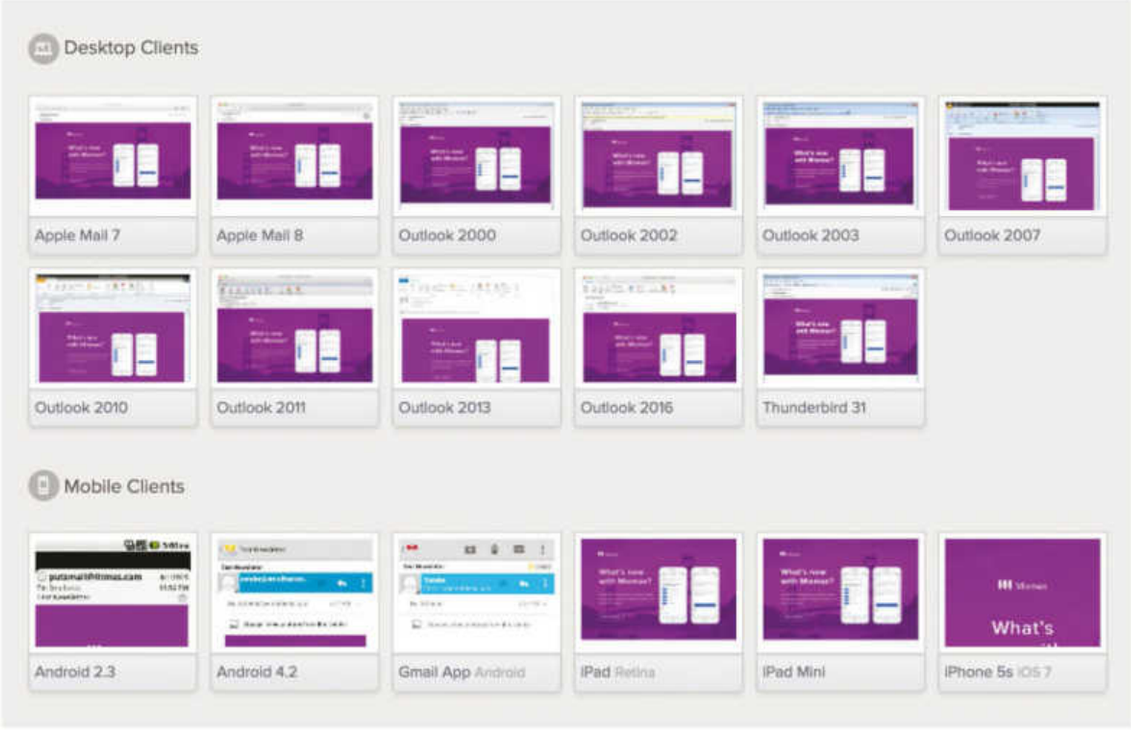
It's a good idea to include an alt attribute or background-color style to your images

between the `<img>` and surrounding `<a>` made that grey border disappear.

```
<span class="logo">
  <a href="http://mochimachine.org/"><img src="http://
mochimachine.org//images/logo.png" alt="MochiMachine
Logo"></a>
</span>
<!-- versus -->
<span class="logo">
  <a href="http://mochimachine.org/">
    <img src="http://mochimachine.org//images/logo.png"
alt="MochiMachine Logo">
  </a>
</span>
```



Thorough testing Style Campaign's impressive 34-piece device library, proving that its advice and posts are battle-tested and accurate



Testing suite The results of a test displayed in Litmus, a comprehensive email testing suite

In the middle of my email design, I noticed that Gmail was causing another problem with my `<a>` elements – it wasn't rendering my links with the custom colour I'd implemented. I remembered that Gmail does not support `<style>` or `<link>` in the `<head>`. Since I had decided to overwrite the default link colour, that meant I had to add a `color` style to each and every `<a>` within my email code.

TESTING, TESTING, TESTING

Where A/B testing your emails is important for determining content and evaluating user interaction, comprehensive testing of email rendering is necessary for ensuring users can even view your emails. Email testing suites show how your emails render across several clients at a glance. All you do is send an email to their system, and they display all the results on one page. They also often cover spam and plain text testing.

Testing suites go a long way in helping you check your email rendering, but not quite far enough. While they typically include at least one version of most email clients, they often miss popular mobile and desktop operating systems, which can heavily affect rendering, even within the same email client. They also take a long time to update with the newest clients.

I highly recommend testing on live devices completely or at least to supplement any email testing suite. You can quickly build up a device library and update faster than an email testing suite might. If you purchase all of your own devices, there's an added benefit of being able to test web pages across mobile browsers, operating systems and devices.

Personally, I own and use a Samsung Galaxy S3 running Android Jelly Bean (v4.3). To my knowledge, no email testing suite includes this version of Android, so I'm often sending myself emails. I test Android Email, Gmail and Inbox on my personal device.

Even Android KitKat (v4.4), which has a market share of 41.4 per cent (as of April 2015, according to Android Developer Dashboard), goes untested in popular email testing suites. These are just a few examples of what you might be missing by only testing with an external source.

ONWARD INTO THE FUTURE

Everything I've talked about in this article will help get you started on your responsive email design journey. My advice here will provide a great foundation upon which to build more complicated emails. But this is just the tip of the email iceberg. To keep up to date on all the latest trends, be sure to do plenty of your own testing, pay attention to the release of new email client versions, and check out the resources I provided.

RESOURCE

Campaign Monitor's CSS Support page for email clients is forever a guiding light in responsive email design: *netm.ag/support-269*

View source files here!

All the files you need for this tutorial can be found at *netm.ag/RWD-271*

ABOUT THE AUTHOR

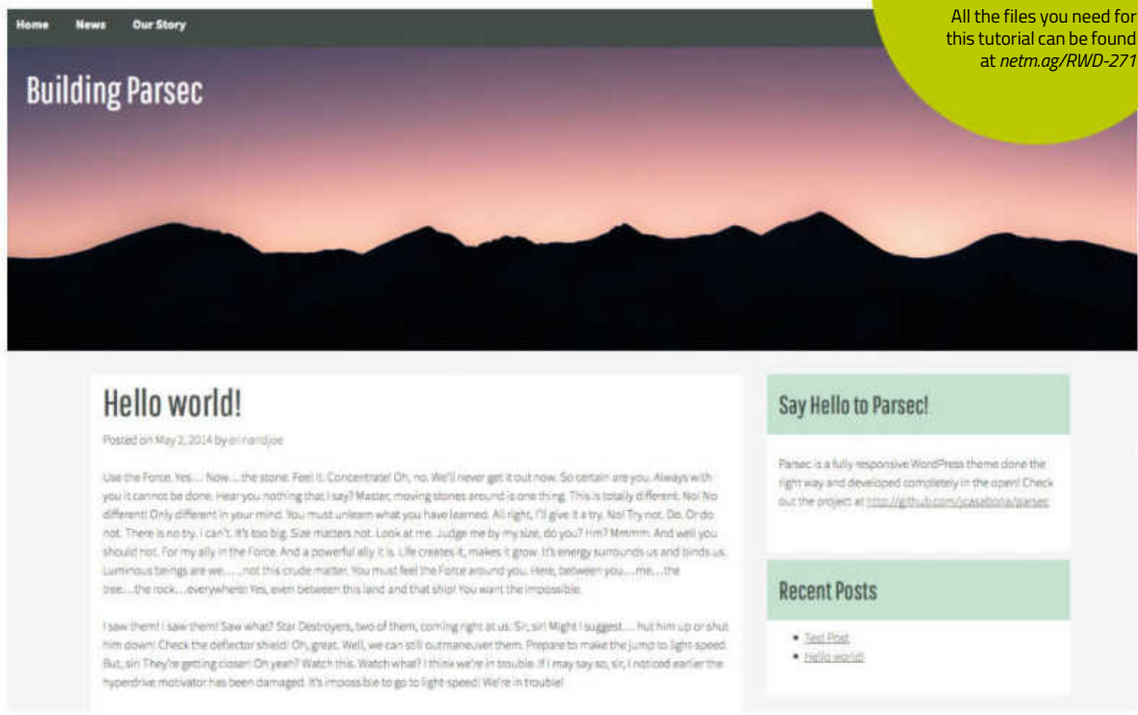
JOE CASABONA

w: *casabona.org*

t: @jcasabona

job: Frontend developer

areas of expertise: HTML, Sass/CSS, PHP, WordPress



WORDPRESS

CREATE A RESPONSIVE WORDPRESS THEME

Joe Casabona walks through the tools and processes he is using to build his reponsive Parsec theme

In 1977, one of the greatest movies of all time came out: *Star Wars: Episode IV – A New Hope*. Most consider it George Lucas' masterpiece. He put a lot of time and effort into creating this movie. He introduced revolutionary special effects and much more, making it a timeless classic.

In 2005, *Star Wars: Episode III: Revenge of the Sith* came out. This is a bad movie. Lucas rushed it, not as much time and effort went into it, and it was shot primarily in front of a green screen. Lucas got lazy and everyone knew it. It doesn't hold a candle to that first movie.

Why am I telling you this? Because there's an important lesson here: if you get lazy, people will know. We also know it's easy to get lazy or feel rushed. 'This has a really tight deadline' sound familiar to anyone else? Last year I gave a talk preaching best practice for responsive design, but I wasn't doing those things myself. I decided to start practicing what I preached.

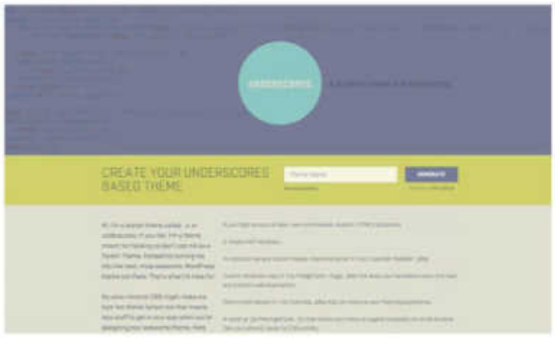
THE GOAL

Since my book *Responsive Design with WordPress* came out in late 2013, a lot has changed. I started using Git on a daily basis. I started using Sass and got introduced to automated tools like gulp. Browsers have advanced quite a bit. For my new template, Parsec (*github.com/jcasabona/parsec*), it was time to update my tools and process (see boxout opposite).

From a personal standpoint, I wanted to improve my process and learn some new tricks. From a deliverable standpoint, I wanted to create a 'good' responsive theme. That means I needed to make a theme that:

RESOURCE

This article by Luke Wroblewski explains why good UIs are obvious, and hiding elements can confuse and put off users: *netm.ag/wrobleski-271*



Underscores I cannot say enough great things about this starter theme. It's perfect for WordPress theme development

- Is mobile-first
- Has content-based breakpoints, not device-specific ones
- Is lightweight and loads quickly

THE TOOLS

First, let's look at the tools. These will change over time. When I started writing my book, I didn't use Sass, and MAMP was my local development environment. I didn't use GitHub for anything besides finished code – it would have been great for my Millennium Flights theme (*netm.ag/flights-271*) and the accompanying plugins.

Using GitHub means readers have the chance to see the development process. It makes it possible to create something, change it and try something new – then to merge the changed version into the finalised implementation. It can also help you write cleaner code. With that in mind, I knew GitHub would be instrumental for Parsec. Plus, by using the wiki and issues features, all the documentation could be made public.

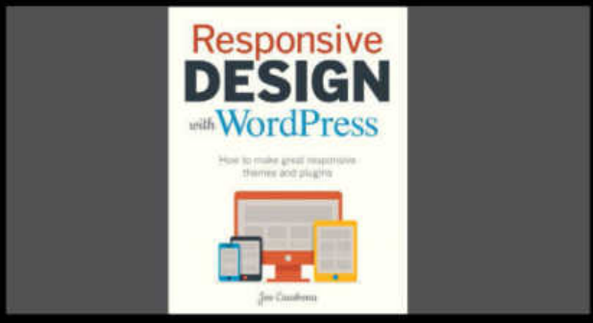
Starter theme Underscores makes just the right number of decisions for you

When developing the Parsec theme, I started with the incredible starter theme Underscores, or _s if you prefer (*underscores.me*). I did this for a few reasons. Underscores has a strong community backing. It's developed by the same folks who develop WordPress' default themes, and it also has about 90 contributors.

Underscores includes code for some common WordPress customisations and well-organised CSS. It makes just the right number of decisions for you. ▶

IN-DEPTH

THE PROJECT



RWD I started writing *Responsive Design with WordPress* over two years ago. While still a good resource, a lot has changed since then

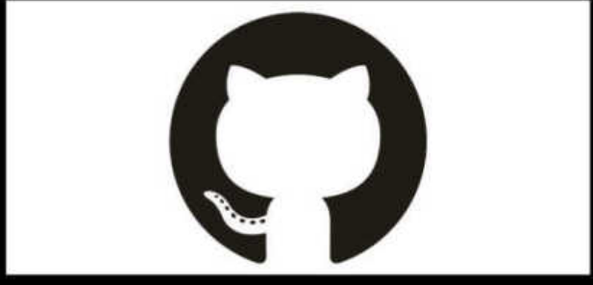
I felt the best way to accomplish the goal I set for myself (namely to improve my process and tackle responsive design for WordPress the right way), was to create a brand new project and document it from planning to launch. The project I decided on is an open source theme called Parsec, along with some related plugins, both custom and downloaded.

I also decided I would do everything from sketches and notes to code and issue tracking, out in the open. GitHub is essential in achieving this. I have my main project branch – the master – as well as a couple of feature branches I'm working on.

On top of that, there are two completely separate project branches that are based on Parsec. I'm adding in certain features that may get merged, but for the most part, these are standalone projects that treat Parsec as a framework.

Within the master branch there are two folders: 'wp-content', and 'assets'. 'wp-content' is what you might expect: the theme and plugins. 'assets' is a folder with any raw assets Parsec may make use of. At the time of the writing, there are Style Tiles and possible fonts.

Finally, GitHub's wiki and issues areas are also put to good use. The issues help me track new features, to-dos and bugs. The wiki includes any resources I have used and may use for reference.



In the open GitHub is a great tool for project management in general, and integral to keeping this project out in the open

It supports enough for two types of layouts and a mobile menu, providing users with a good starting point without having to undo anything.

Finally, Underscores supports Sass, another tool I use heavily. Sass is an incredible piece of software. It allows you to build extensible, flexible CSS. Since I'm using Parsec as a starting point for my themes, my code is organised in such a way that I should be able to swap colours and fonts easily. Sass helps with that.

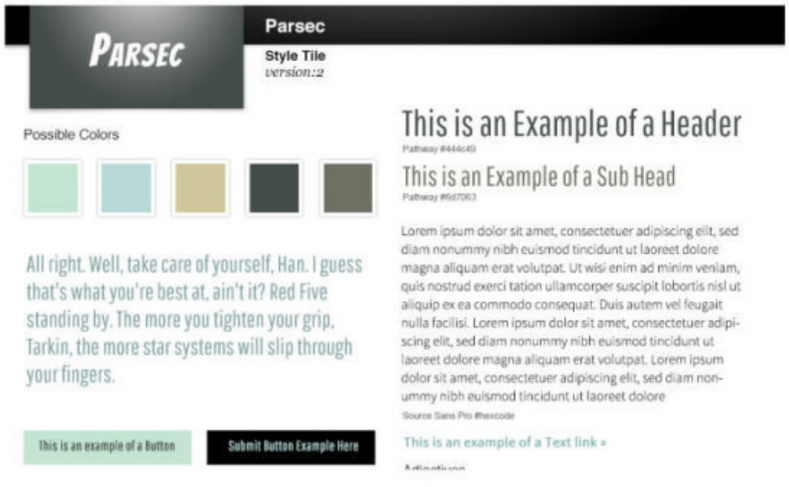
THE PROCESS

Whenever I design a website, I follow the same steps. Let's take a look at those now:

0 Choose the content
1 Sketch for narrow and wide widths
2 Create a Style Tile
3 Code
4 Test

In true programmer fashion, I started this list with a zero. My justification is that content should be ready before the design process starts. This is something I tell everyone. It's also something I do for all my projects. I know expecting this for every project sounds like an 'in an ideal world' situation, but it's so important!

I explain the importance of starting with content by asking a simple question: 'How can I know what is most important if the content isn't finished?' At the start of the process, knowing how the content stacks is integral. This is especially true when designing mobile-first. Not only does it indicate to me the purpose of the website and what is most important to users, it also prevents me from adding extra elements. I'm looking at you, mega-menus, parallax and image carousels. Once the content is ready, we start the process.



The dark side This is version two of my Style Tile for Parsec. In my final version, I made the colours lighter and added some cleaner fonts

SKETCH

To me, sketching is a must-have part of the design process. This stage allows me to lay out the content and see how things might look before putting any code on the page.

It also forces me to consider some important questions. Parsec is intended to be a basic, almost starter theme, so early on I needed to figure out what exactly I want to use it for. Would it be primarily for personal sites? Business sites? Blogs, wedding sites or event sites? As I started to sketch, I realised I didn't know the answer to this question.

Through sketching, I was able to illicit some answers. I'm making the theme for myself first, so what sites do I need to make? My current personal site uses WordPress' default theme, so that could use a facelift. My wedding is coming up. That will also need a website.

In the end, I decided Parsec would be a personal site theme. That said, if you look at the GitHub repo, you will see I was able to create a wedding site and an events site using the same base theme.

Sketching is the freest form of design in this medium. It means you can start with a blank slate, not confined by the tools you're using – whether they be code, Photoshop, Sketch or some other programme. You aren't limited by the assumptions you'd make in front of a computer screen.

STYLE TILES

One drawback of sketching is that you can't get a good idea of how elements will mesh together. For this, determining fonts and font sizes, colours and other graphics is necessary. That's where Style Tiles come in.

A big focus of my design process, and something I've been encouraging others to do for a while is to avoid using Photoshop. When you're designing websites with responsive, fluid layouts, using a set Photoshop canvas can be detrimental. How do you decide what widths to design at? Are you going to make three or four different mockups for the different widths, showing how – but not when – the layouts should change? In my opinion, going from sketch to code or some other live layout updating program like Macaw is a better option.

But still, you may want to pair colours, fonts and graphics to see if they play well with each other. Style Tiles (*styletil.es*) is a nice substitution for Photoshop, and allows you to communicate a design's visual brand.

I use Style Tiles to play with colour schemes, choose font faces and sizes, and see how certain graphics look. I am able to iterate a lot quicker when I don't have to worry if certain font pairings will

work, or how well the colours I chose play together. It's thanks to Style Tiles that I ended up with the nice light and clean colour scheme on Parsec instead of the darker version I came up with first (both Style Tiles are in the GitHub repo).

Once the Style Tile is complete, I use it as a reference for the design, first setting up my Sass variables. This is another fantastic advantage to using both Style Tiles and some kind of CSS preprocessor – the Style Tile translates nicely into Sass variables.

START CODING

With sketches and styles in hand, it's time to put code to editor. Since Underscores gives us a bit of a head start, the first thing I do is define Sass variables and see what the page looks like out of the box.

First, I define some base colours, then create element-specific variables based on those colours:

```
$color-primary: #333333; //dark grey
$color-secondary: #C05300; //burnt orange
$color-accent-two: #006776; //teal
...
$color-background-body: $color-white;
$color-background-screen: lighten($color-gray-base, 36%);
```

Sketching means you can start with a blank slate, not confined by the tools you're using

After that, I assign some variables for font families and sizes, breakpoints, padding and more. This is all part of the 'structure' of the site, helping keep consistent vertical rhythm and improving the design. Adding breakpoints as variables is especially helpful. You can quickly change or add more based on your content.

The last piece I do for setup is some common mixins. I won't share all them here, but I do want to point out two helpful ones:

```
@mixin breakpoint($breakpoint, $direction: min) {
        @media (#{$direction}-width: $breakpoint)  {
                @content;
        }
}

@mixin breakpoint-tweener( $bp-start, $bp-stop ) {
        @media screen and (min-width: $bp-start) and
(max-width: $bp-stop) {
                @content;
        }
}
```

Engagement levels Luke Wroblewski notes that after Polar's navigation changed to a less obvious design, engagement plummeted

The first is your standard breakpoint mixin. You pass an em value and either `min` or `max`, and the media query gets built for you.

The second is what I call a 'breakpoint tweener' – or styles that belong only within two specific breakpoints. It's not too often I use this, but often enough to warrant a mixin for it.

IMPLEMENTING NAVIGATION

Earlier I mentioned that Underscores has a nice way of implementing responsive navigation: you tap a 'menu' button to reveal the hidden navigation. I changed this to a hamburger icon, and I will likely optimise it when I implement gulp.

One note: earlier this year, Luke Wroblewski wrote a great article (*netm.ag/wrobleski-271*) about how obvious is better when it comes to UI. The gist is that hiding elements with an icon might look nicer, but it negatively affects engagement. The word 'menu' for example, is more obvious than the hamburger icon, even if the latter seems ubiquitous.

IMPLEMENTING IMAGES

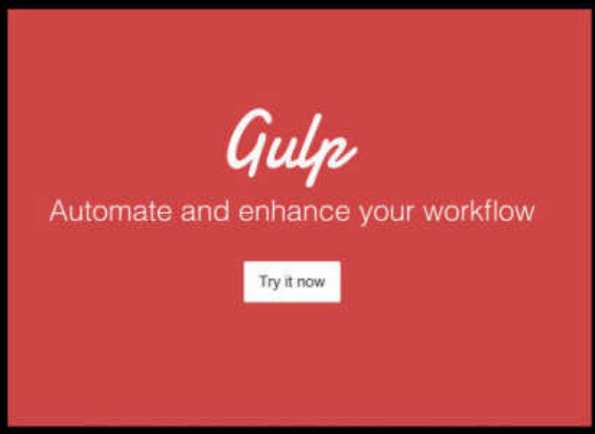
There are several ways to do responsive images. I'm using a double-pronged approach: multiple versions of images in CSS, and the `<picture>` element inline. First the CSS:

```
.impact-img {
  background: url( $asset-path + 'img/impact-sm.jpg' ) center
no-repeat;

  @include breakpoint( $breakpoint-1 ) {
    background-image: url( $asset-path + 'img/impact-med.
jpg' );
  }
```

FOCUS ON

FUTURE WORK



Asset size gulp will serve as a great tool to help reduce the size of assets, from JavaScript to images

There are a few things that, at the time of this writing, had not been completed with Parsec. I didn't include them in the main body text because I wanted to speak about my experience so far, but here's what I have planned for the future.

Something I plan to implement is gulp. gulp is one of many JavaScript-based automation tools that can perform tons of great tasks – including basic ones like compiling Sass, and combining and minifying JavaScript. It can also perform powerful tasks like creating SVG sprites and icon fonts on the fly, or connecting with Git for deploying code. This tool in particular will go along way in optimising Parsec's code and reducing load times.

In line with a point that Luke Wroblewski made about obvious design being better, I plan to change the hamburger icon currently serving as the menu button back to the word 'menu' in a future iteration. This may allow me to eliminate font icons completely in the base design, while also providing a better user experience.

On the server side, I want to implement a solid RESS (REsponsive design with Server Side detection) function, making use of `wp_is_mobile()` as well as some sort of API, like Scientiamobile. This will allow me to dynamically determine a device and serve up assets accordingly. This means I will only need to print one type of navigation – or perhaps I will be able to exclude some JavaScript fallback because I know the user's device supports a feature natively. The possibilities are endless. In the end, we should have a rock-solid responsive WordPress theme!

```
  @include breakpoint( $breakpoint-3 ) {
    background-image: url( $asset-path + 'img/impact.jpg' );
  }
}
```

As you'll see, I use smaller images to start, and as the screen gets bigger, I load in bigger images. While not a perfect assumption, I decided that if you're on a smaller screen, you are likely on a mobile device. In this case, I use an image that will load faster and doesn't use as much data. The same sort of logic is used for the `<picture>` element.

A great piece of news came out around June of 2014: the `<picture>` element had become an official part of the HTML spec, meaning we could start using it! This element allows us to define more than one image source, then define breakpoints to display each image:

```
<picture>
  <source srcset="extralarge.jpg" media="(min-width: 1000px)">
  <source srcset="large.jpg" media="(min-width: 800px)">
  <img srcset="medium.jpg" alt="This is the alt tag, and default image">
</picture>
```

For RWD, there are three major testing areas: browser, device and connectivity

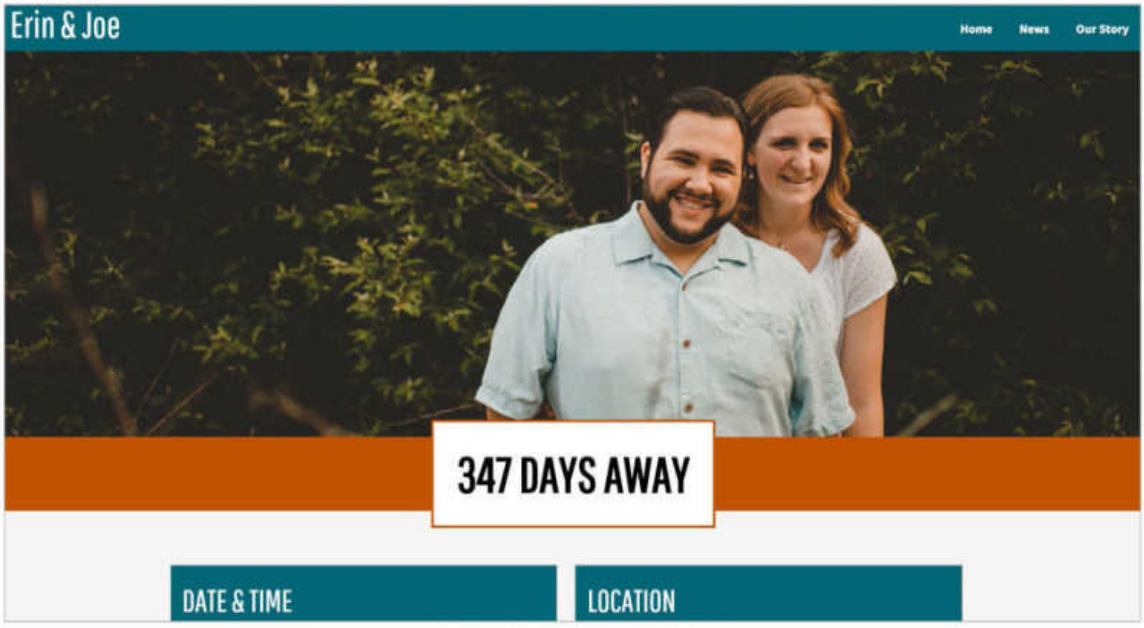
The only problem is that it's not supported by all browsers yet. Luckily there is picturefill.js and a fantastic plugin called RICG Responsive Images (*netm.ag/plugin-271*) to help with this. After writing my own implementation for responsive images, I decided this plugin was worth a look.

At the time of this writing, I am still testing it. You can see my implementation in the wedding branch of the project.

TESTING

I want to close out this article with some tools I'm using for testing. I want to make it super-clear that no matter what, you should test on actual devices. First, let's talk about how you should test. For responsive design, there are three major areas of testing: browser, device and connectivity.

Browsers are what we're most familiar with, and responsive design didn't make things any easier for us. On top of desktop browsers, we now need to test



Parsec fork My wedding website is a fork of Parsec that includes some nice features and styles, proving Parsec's flexible base structure

mobile OSs and the browsers they use. This non-comprehensive list includes:

- Safari on iOS
- Chrome on iOS
- Firefox on iOS
- Chrome on Android
- Browser on Android
- Firefox on Android
- IE for Windows Phone

Of course, the browsers you choose to support depends on your user base. In the case of Parsec, the more support, the better.

Then there is devices. Part of the reason for the content- versus device-based breakpoints debate is the sheer number of different screen resolutions around (over 6,000 on Android alone, last time I checked). Different mobile OSs also handle certain things differently.

You need to determine which devices you should test, but I usually recommend:

- The latest and previous versions of the iPhone
- The iPhone 6+
- The latest and previous versions of Google's Nexus device
- An Android 4.0 device
- The iPad and iPad Mini
- A 7" Android Tablet
- A 10" Android Tablet
- A Windows Phone

This can be tough, because you may not have these devices at your disposal. There are a few things you can do. I use BrowserStack, which works well, but nothing substitutes actual devices. Adobe EdgeInspect gets you part of the way there, as it will allow you to simultaneously test any devices you own at the same time.

You can try find a device lab (*opendevicelab.com*) or ask some friends if you can test on their devices. If worst comes to the worst, you could always walk into your local AT&T store and load your site on the devices there. Efficient testing and free advertising!

Finally, you should test connectivity and loading times. Try testing on Wi-Fi, 4G and 3G at a minimum. Slower if you can. Do this while moving and standing still – you will get a much better idea for how fast your site loads, allowing you to adjust accordingly. Remember, 80 per cent of users will abandon a site if it takes more than five seconds to load.

FINISHED PROJECT

I launched Parsec in late 2015, and you can see it in action at *casabona.org*. I plan to continue to work on it to develop it further – you can watch my progress at *github.com/jcasabona/parsec*.

I had lofty goals and an aggressive timeline for this project. But I also wanted to put the proper effort into it, test it as much as I could, and do things the right way. I want this to be an *Episode IV*, not an *Episode III*.

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/wordpress-254*

ABOUT THE AUTHOR

JOE CASABONA

w: *casabona.org*

t: @jcasabona

areas of expertise:
HTML, CSS, PHP, MySQL, WordPress

WORDPRESS

BUILD A RESPONSIVE WORDPRESS PORTFOLIO

Joe Casabona explains how to create a plugin to set up a new Custom Post Type and a new theme template, for your own responsive portfolio

Web development may change rapidly, but two things that are here to stay are WordPress and responsive design. Knowing how to build responsive WordPress themes and plugins is a must. In this tutorial, we will look at building a WordPress plugin and theme template for a responsive portfolio.

We will work with two mocked-up templates: an archive page, which will list all of the recent projects, and a single page, which will show a specific project. The archive page for the portfolio is a pretty simple one with a header and three columns of projects at full width. This will shrink to two columns, then one column, as the screen gets smaller. The HTML and CSS is available at GitHub: *netm.ag/wordpress-254*. Each project on the page will have this structure.

This is the HTML that will be generated by the WordPress Loop:

```
<div class="card">
    <img src="project.png">
        <h3>Name of Site</h3>
        <p>Short description and a link <a href="#">read more...</a></p>
</div>
```

The single page is going to have a similar layout, wrapping all of the text in a container called `.project` instead of `.card`. The CSS is also going to be fairly lightweight and scaled. You may notice in the site files there is a `style.scss` file. I've developed all

VIDEO

Joe Casabona has created an exclusive screencast to go with this tutorial. Watch along at: *netm.ag/video-rwd-254*

the CSS using Sass, but fear not: the generated `style.css` is compiled in a readable way.

CREATING A NEW CUSTOM POST TYPE

A Custom Post Type is an object in WordPress that allows us to add any types of content we want to the WordPress editor, treating them the same way as posts. In a fresh WordPress install, there are menu options for Posts and Pages, each of which handles content differently. With Custom Post Types, we can add options for creating new types of content to the WordPress admin menu. We'll create a Custom Post Type called Portfolio.

We're going to develop this Custom Post Type as part of a bigger WP plugin for portfolio projects. While we could add the functionality to the theme, this is bad practice because then our content is tied to our design: if we change the theme, we lose the portfolio. We will handle display through two

RWD is here to stay. Knowing how to build responsive WordPress plugins is a must

methods: templates/template tags, and shortcodes that can be used through the editor.

The first step is to define the plugin. Create a folder in '/wp-content/plugins/' and name it whatever you like. I've named mine '/jlc-projects/'. Inside that folder, create a file of the same name (for example, 'jlc-projects.php') and add this code:

```
<?php
/*
Plugin Name: Joe's Portfolio Plugin
Plugin URI: https://github.com/jcasabona/wp-portfolio
Description: A simple plugin that creates and display a
projects portfolio with WordPress using custom post types!
Author: Joe Casabona
Version: 1.0
Author URI: http://www.casabona.org
*/
define('JLC_PATH', WP_PLUGIN_URL . '/' . plugin_basename(
dirname(__FILE__) ) . '/' );
define('JLC_NAME', "Joe's Portfolio Plugin");
require_once('jlc-project-cpt.php');
?>
```

There are a few things going on here. The first is the standard plugin definition for a WordPress plugin; the next few lines create constants and then include the ▶

FOCUS ON

LEARNING SASS

Sass (*sass-lang.com*), which stands for Syntactically Awesome Style Sheets, is a CSS preprocessor that lets you write CSS in a way similar to programming languages. You can define variables, make comments hidden from the output file, write functions and extensible CSS classes, perform mathematical operations, and more.

As someone who primarily considers himself a programmer, I find this method of writing very appealing. I can abstract away a lot of what I would normally repeat. Plus, as far as RWD goes, Sass makes it much easier to keep all of the media queries for one class together. For example:

```
.card{
        /* Default Styles Here */
        @media screen and (min-width: $bp-medium){
                display: inline-block;
                width: 40%;
                }
        @media screen and (min-width: $bp-large){
                width: 44%;
                }
        }
```

No matter what size your project is, I would recommend using Sass for the CSS. It will make managing your style sheets a lot easier, and make responsive sites better, through improved organisation and by compiling multiple resources into one.

If you want to learn Sass, the best resource out there is *Sass for Web Designers* by Dan Cederholm (*netm.ag/Sass-250*).

Book apart Dan Cederholm's *Sass for Web Designers* is an excellent Sass primer

FOCUS ON

RWD AND WORDPRESS

This article only scratches the surface of responsive design using WordPress. There are a lot of other factors to keep in mind. Remember that developing responsive websites is about more than just the site shrinking on small screens. You should also ensure that you're using best practices by:

- **Using ems for breakpoints**

These are much more accessible for users who may have different browser settings and handle the wide variety of screen resolutions much better.

- **Using content-based breakpoints**

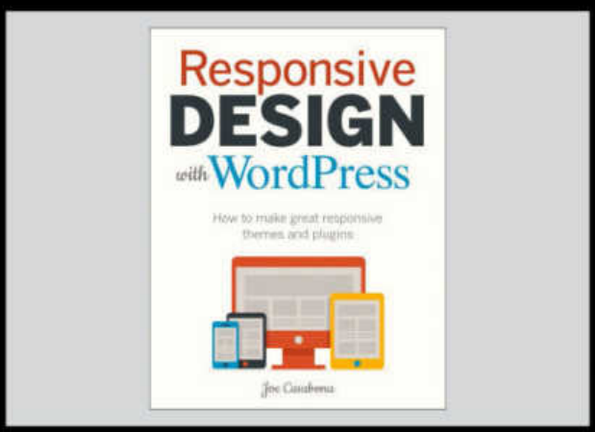
Make sure you're developing in such a way that your breakpoints are determined by the content of your website and not the device's resolution. That way you know your content looks good universally, not just on three specific devices.

- **Keeping speed and size in mind**

In a perfect world, everyone would be on Wi-fi or 4G and not have to worry about data caps. Unfortunately, that isn't the case. As we develop websites, we need to remember to load only what's needed, reduce page weight and remove extra HTTP requests. This will make things much easier on the user.

Tools like Sass and WordPress can help us achieve these goals, especially the final one. In my recent book, *Responsive Design with WordPress* (*rwdwp.com*) I go over best practices for responsive design, why they are important and how to implement them in WordPress, as well as pitfalls to avoid.

I also include several tutorials like this one in the last chapter, taking you through developing forms, photo galleries, and even a products page responsively in WordPress.



Further reading Joe Casabona's book complements and extends this tutorial

files the rest of the plugin needs. At this point, there is only one other file: 'jlc-project-cpt.php'.

You will also notice that I'm using the prefix 'JLC_' (or 'jlc-') for everything. You should choose your own prefix to use. Prefixing variables and function names will decrease the chance of your plugin conflicting with other themes or plugins.

Before we jump into 'jlc-project-cpt.php', I want to add one more bit of code to 'jlc-projects.php'. The code below will create a new image size, which we will use with our Custom Post Type:

```
if ( function_exists( 'add_theme_support' ) ) {
    add_theme_support( 'post-thumbnails' );
    add_image_size('jlc_project', 1100, 640, true);
}
```

Now it's time to create 'jlc-project-cpt.php'. I'll only be discussing the important code here, but you'll find the complete code on the GitHub repo. First (after the opening `<?php` tag) we define the Custom Post Type:

```
add_action('init', 'jlc_projects_register');
function jlc_projects_register() {
    $args = array(
        'label' => __('Portfolio'),
        'singular_label' => __('Project'),
        'public' => true,
        'show_ui' => true,
        'capability_type' => 'post',
        'hierarchical' => true,
        'has_archive' => true,
        'supports' => array('title', 'editor', 'thumbnail') ,
        'rewrite' => array('slug' => 'portfolio', 'with_front'
        => false)
        );
    register_post_type( 'portfolio' , $args );
    register_taxonomy("jlc-project-type", array("portfolio"),
    array("hierarchical" => true, "label" => "Project Type",
    "singular_label" => "Project Type", "rewrite" => true));
}
```

This is your standard Custom Post Types definition function. We add an action to call it on `init`, then send

New UI The admin page to add a new project. Notice the Project Link section

our list of arguments to `register_post_type()`, along with the type's slug, which will be 'portfolio'. After registering the post type, we register the custom taxonomy to go along with it. It's important to keep these two functions together. If you don't, and the taxonomy somehow gets registered first, WordPress will throw an error.

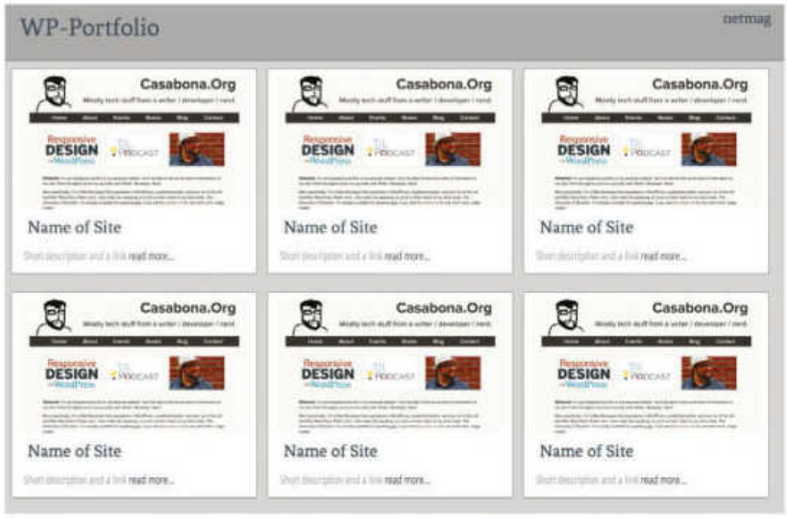
After the Custom Post Type is defined, it's time to add the custom metadata we want to use. Our Custom Post Type supports a title, the editor (which will serve as the body text), and a thumbnail, which is where the featured image will go. There is one more thing I like to add to my portfolio pieces: a URL to the website I'm showcasing. First, we'll create the function that will add this box in the admin:

```
add_action("admin_init", "jlc_projects_admin_init");
function jlc_projects_admin_init(){
    add_meta_box("jlc-projects-meta", __("Project Link"),
    "jlc_projects_options", "portfolio", "side", "low");
}
function jlc_projects_options(){
        global $post;
        if ( defined('DOING_AUTOSAVE') && DOING_
        AUTOSAVE ) return $post_id;
        $custom = get_post_custom($post->ID);
        $link = $custom["jlc_projects_link"][0];
?>
  <input name="jlc_projects_link" placeholder="http://"
  value="<?php echo $link; ?>" />
<?php
    }
```

These functions are fairly straightforward. When the admin is initiated (that is, loaded), we'll call a function called `jlc_projects_admin_init()` that will create a new meta box for portfolio items. In order to generate that box, a new function, `jlc_projects_options()`, is called.

Once inside the `options` function, we simply grab the `link` value, which we've called `jlc_projects_link`, and print it out. But first, we want to make sure an autosave isn't being performed. If it is, we will probably lose data. After that, we need to actually save the metadata when the post is saved:

```
add_action('save_post', 'jlc_projects_save');
function jlc_projects_save(){
    global $post;
    if ( defined('DOING_AUTOSAVE') && DOING_AUTOSAVE ){
        return $post_id;
    }else{
        update_post_meta($post->ID, "jlc_projects_link",
        $_POST["jlc_projects_link"]);
    }
}
```



Basic grid The HTML template used here, at full width. It's a simple header with three columns of projects

With the admin section for our Custom Post Types created, it's time to add some frontend functionality to help display our projects to visitors. This consists of an archive template, a single page template and a shortcode (not covered in this tutorial). But before we do that, there is one other function we're going to create for displaying images: picturefill.js.

This piece of JavaScript (GitHub repo at *rwdwp.com/23*) allows you to define a set of media queries to switch an image to a version friendlier to the size of the screen it is being viewed on. This also has implications for load time, as you can probably assume that a smaller screen means a mobile device using 4G, 3G, or even EDGE. I know that isn't always the case, but it's better than nothing.

You can see the markup pattern for a standard picturefill element on the GitHub repo. We can have an unlimited number of `<span>` elements for each size of the image we have. There is also a fallback for users without JavaScript. As you can imagine, since WordPress creates multiple versions of every image we upload using the Media Uploader, it lends itself nicely to picturefill.js.

The first thing we should do is load the script, which is in the '/js/' folder in our plugin's directory. We add the following code to 'jlc-projects.php':

```
function jlc_projects_scripts() {
    wp_enqueue_script( 'picturefill', JLCP_PATH.'js/
    picturefill.js', array());
}
add_action( 'wp_enqueue_scripts', 'jlc_projects_scripts' );
```

This will load our JavaScript with the scripts being loaded by other plugins. It will also ensure that we aren't loading picturefill.js more than once.

RESOURCE

Scott Jehl's original blog post about picturefill.js discusses the need for the polyfill and includes a link to a demo site: *netm.ag/picturefill-254*

As our projects will be using the featured image section to display the screenshot, we can replace the default featured image markup by using the `post_thumbnail_html` filter. Note that this function will replace all featured image sections on the site. If this causes a conflict (it probably will), you should add some conditionals to your plugin to make sure this filter is only being used on portfolio pages.

```
add_filter( 'post_thumbnail_html', 'jlc_projects_get_
featured_image');
function jlc_projects_get_featured_image($html,
$aid=false){
    $sizes= array(''thumbnail', 'medium', 'large', 'jlc_project',
    'full');
        $img= '<span data-picture data-alt="'.get_the_
        title().'">';
        $ct= 0;
        $aid= (!$aid) ? get_post_thumbnail_id() : $aid;
        foreach($sizes as $size){
            $url= wp_get_attachment_image_src($aid,
            $size);
            $width= ($ct < sizeof($sizes)-1) ? ($url[1]*0.66) :
            ($width/0.66)+25;
            $img.= '
                <span data-src="'. $url[0] .'"';
            $img.= ($ct > 0) ? ' data-media="(min-width: '.
            $width .'px)"></span>' :'></span>';
            $ct++;
        }
        $url= wp_get_attachment_image_src( $aid,
        $sizes[1]);
    $img.=  '<noscript>
        <img src="'.$url[0] .'" alt="'.get_the_title().'">
                </noscript>
            </span>';
        return $img;
}
```

Mobile view A portion of the archive template displayed on a mobile-sized screen. The cards shrink to single column with centered images

There are a few things going on here. First, the function has an array of all the image sizes in WP that we want to use. If you have your own sizes defined, you need to add them. This is so the picturefill element is accurately populated.

After some set-up (defining the image sizes, opening the picturefill element, initialising a counter), it moves through the `$sizes`, printing an image entry for each. For each entry, `wp_get_attachment_image_src()` is called to grab the URL of the image based on the image's ID (which `get_post_thumbnail_id()` returns based on the post ID) and the size. `wp_get_attachment_image_src()` returns an array that includes the image, the width, the height, and whether or not it's cropped.

There is also a bit of maths going on here, to calculate when to determine the breakpoints, as well as how to handle the thumbnail image. I'm not going to discuss this in detail here, but it's worth noting that this is an important problem to solve. Now any time we get the post's thumbnail, the HTML returned will be from our function.

If your plugin includes CSS, keep it minimal so it doesn't butt heads with the main theme

CREATING THE ARCHIVE PAGE

Next, we will create the archive template for the new Custom Post Type. This is what will be displayed by default and will serve as our main portfolio page. Note that we will not be creating the site's homepage in this tutorial, but doing so would require either a template tag or shortcode that will execute a custom Loop using `WP_Query`.

Create a new file in whatever theme directory you are using and call it 'archive-portfolio.php'. WordPress's template hierarchy is smart enough to know that, when a user is on the portfolio page, it should display the content using this template. I recommend copying 'page.php' for this template. We will simply replace the Loop portion.

I recommend that you use a template without a sidebar, or a single-column template. The CSS referenced here will work a bit more nicely. Here's what our Loop looks like:

```
<?php while (have_posts()) : the_post(); ?>
    <div class="card">
        <?php the_post_thumbnail('jlc_project'); ?>
        <h3><?php the_title(); ?></h3>
```

```
        <p><?php echo get_the_excerpt(); ?> <a href="<?php
        the_permalink(); ?>">read more...</a></p>
    </div>
<?php endwhile; ?>
```

This should be pretty straightforward. We are replacing the default HTML for `the_post_thumbnail()`, so we don't need to worry about which image to use because all sizes will be returned using picturefill.js markup. I opted to use `get_the_excerpt()` in order to exclude any markup included by `the_excerpt()`.

When designing a plugin that includes some CSS, it's important to make it as minimal as possible so it doesn't butt heads with the theme's CSS or give the user the ability to exclude your CSS completely. As we're creating templates within the theme, we have a little more wiggle room.

Here's a portion of the (Sass-generated) CSS that I've added to each project on the archive page:

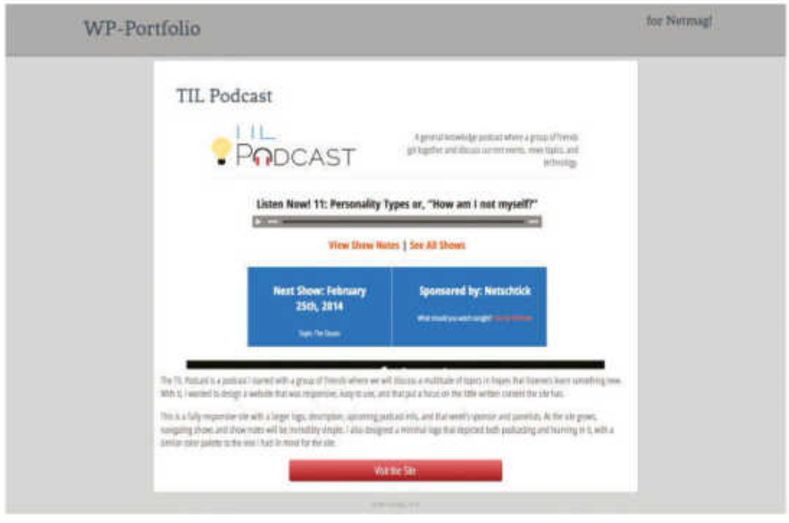
```
.card img {
    display: block;
    margin: 0 auto; }
@media screen and (min-width: 45.88em) {
    .card {
        display: inline-block;
        width: 40%; } }
@media screen and (min-width: 54.62em) {
    .card {
        width: 44%; } }
@media screen and (min-width: 76.38em) {
    .card {
        width: 29%; } }
@media screen and (min-width: 99.4em) {
    .card {
        width: 30%; }
```

I've determined which breakpoints were best for placing the project cards side by side. I've also made the featured images centre automatically.

CREATING THE SINGLE PAGE

Now we'll create the single template for portfolio projects. Whenever a user visits a single project's page, this is what will display. Create a new file in your theme, call it 'single-portfolio.php' and copy another template in there (I'd recommend whatever you used for 'archive-portfolio.php'). This time we will be replacing the Loop with this code:

```
<?php while (have_posts()) : the_post(); ?>
    <h2><?php the_title(); ?></h2>
        <?php the_post_thumbnail('jlc_project'); ?>
        <?php the_content('Read the rest of this entry'); ?>
        <?php
```



Single project The single project view. The project reduces in width as the page grows to keep the text easy to read

```
            if(function_exists('jlc_projects_get_link')){
                $jlc_link= jlc_projects_get_link($post->ID);
                if($jlc_link){
        ?>
                <a href="<?php print $jlc_link; ?>"
                class="button">Visit the Site</a>
        <?php
                }
            } ?>
<?php endwhile; ?>
```

This looks similar to the archive template, but we call an extra function: `jlc_projects_get_link()`. We will add this to our plugin, and it will return the URL for the current project. In the event there is no URL, false should be returned and no button is displayed.

Here's what the function (located in `jlc-projects.php`) looks like:

```
function jlc_projects_get_link($id){
    $url= get_post_custom_values('jlc_projects_link', $pid);
    return ($url[0] != '') ? $url[0] : false;
}
```

The CSS for this page will depend largely on the theme: I've used some CSS to generate a nice button. Make sure that whatever CSS you create yourself does not interfere with the main theme.

IN CONCLUSION

By now, we've created a plugin to add a new Custom Post Type for portfolios, integrated picturefill.js to handle images better, and created two theme templates to display the information.

The GitHub repo for the tutorial contains all of the code shown here, plus the theme I used, a shortcode and a template tag.

BOOK

The issues raised in this tutorial are discussed at greater length in Joe Casabona's book, *Responsive Design with WordPress*: *rwdwp.com*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/modularGit-261*

ABOUT THE AUTHOR

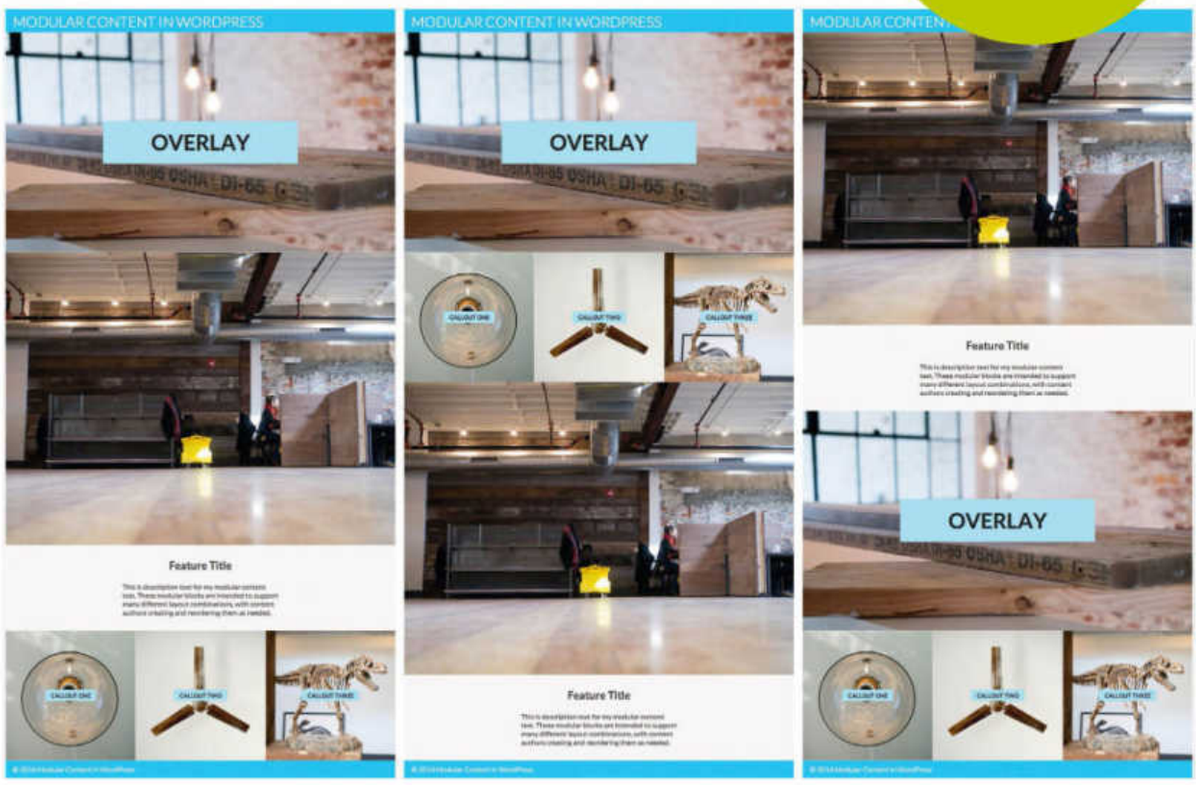
MARK LLOBRERA

w: *bluecadet.com*

t: @dirtystylus

job: Senior developer, Bluecadet

areas of expertise: HTML, JavaScript, WordPress, Drupal



WORDPRESS

BUILD MODULAR CONTENT SYSTEMS IN WORDPRESS

Mark Llobrera walks through how to use Advanced Custom Fields to create a WordPress CMS that is flexible as well as structurally sound

The key word that has informed my work for the last few years has been 'flexible'. A website needs to be flexible for the end user, morphing to meet them on their chosen device. But our content authors need flexibility, too, in how they create content for the site. They need to be able to mix different kinds of content to create a compelling site.

One of the ways we talk about this flexibility for both users and authors is using the term 'modular content'. Loosely defined, modular content means breaking down our content into smaller chunks that can be combined in many different ways. It is not a new idea, but it is one that is given new urgency by our new multi-screen, multi-device web. A 'page' is simply too big a unit – it must be broken down into smaller components to enable it to bend and change to better serve its authors and users.

WordPress has a few options that can be used to add some of this flexibility into a CMS. Elliot Condon's Advanced Custom Fields (*advancedcustomfields.com/pro*) has long been an essential plugin for extending the capabilities of WordPress in order to build custom CMSs. The plugin has two features that are well suited to our goals of creating a CMS that supports our flexible design

VIDEO

Mark Llobrera has created a screencast to accompany this tutorial. Watch along at *netm.ag/modularVid-261*

system, namely the Flexible Content Field and the Repeater Field.

These features allow designers and developers to create modules and craft markup to represent those modules, but they also give content authors flexibility in how they order those modules. So instead of needing many different page templates, we can use a smaller number of flexible content types and templates that are able to render a wide range of possible pages.

IDENTIFYING OUR MODULES

Let's start with an example: a product details page template. A common scenario is to have a large, evocative image at the top, supported by a section (or sections) highlighting individual features (with an optional image). Finally, callout blocks to related products could follow.

We can start by identifying our design system. There are a number of ways to do this, but at the moment I like to sketch out my site's possible templates and what they will need to support. Once I've done that, I start circling common visual

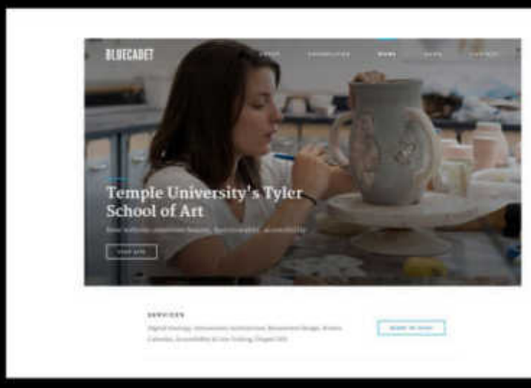
A website needs to be flexible for the user, morphing to meet them on their chosen device

structures that can support those content modules. From there I can move on to wireframes that clearly identify those modules.

In the scenario shown in the image on page 221, the template calls for three modules: A) a full-width image with text layered on top, B) a centered image with text running underneath it, and C) a series of callout blocks that have a thumbnail image with text layered on top. Our goal is a template that can support any of those blocks, in any order: ABC, ACB, BAC, BCA, ABBC and so on. Certain orders will make more sense given the desired hierarchy of the page, but we want to be able to support all options.

CREATING MODULES IN WORDPRESS

Now that we have an idea of what kinds of content we'll need to support, we can start creating our modules in WordPress. We will use Advanced Custom Fields Pro (*advancedcustomfields.com/pro*), a paid plugin that includes the Flexible Content Field and Repeater Field. Once you have the ACF Pro plugin activated, go to the new menu item `Custom Fields > Custom Fields` and add a new `Field Group`. Call it 'Core

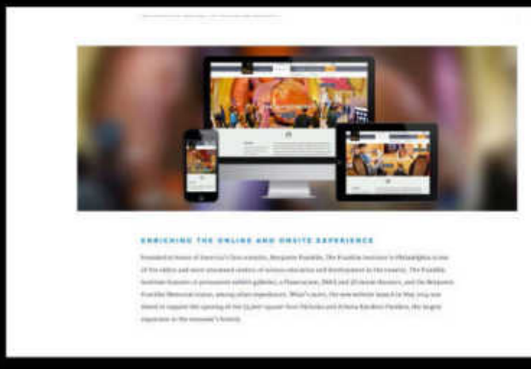


CASE STUDY

OUR TOUGHEST CLIENT: OURSELVES

At Bluecadet we've had a lot of experience building modular content systems for our clients. Our toughest client this year, however, might have been ourselves. We entered this year with the goal of redesigning our website (*bluecadet.com*), and it took a lot of hard work to create a design system that could accommodate the range of our work. For our case study pages we needed support for varying numbers and combinations of text, video and imagery. Additionally, we needed the system to be intuitive for our teammates who would be entering and maintaining content.

We ended up creating a design system with modular content blocks, including: a full-width single image, an image slideshow, a video player, a video loop, a pull quote, and regular text. Once we started building in WordPress, Advanced Custom Fields' Flexible Content Field allowed us to create each content block as a separate layout, all shared by one post type and one template file. Our content authors appreciated the ease of creating and re-ordering the different module layouts, which let them put together the right layout for each project case study page.



Mix'n'match Content authors create layouts from a range of content modules

RESOURCES

FURTHER READING

The overlap between modular content and frontend style guides is very large, and Anna Debenham has written a good overview on different approaches to building them, called 'Front-end Style Guides' (*netm.ag/style-261*). She also maintains a collection of frontend style guide resources (*netm.ag/gimme-261*).

Dave Rupert of Paravel Inc. wrote 'Responsive Deliverables' (*netm.ag/deliverables-261*) which deals with design systems, style guides and different client deliverables.

I've focused on WordPress's Advanced Custom Fields, but ExpressionEngine and CraftCMS both offer the Matrix field (*buildwithcraft.com/features/matrix*), which behaves very much like the ACF Flexible Content Field.

Karen McGrane's book *Content Strategy for Mobile* (*netm.ag/content-250*) and Sara Wacher-Boettcher's book *Content Everywhere* (*netm.ag/content-237*) both deal with identifying and structuring content, which is the starting point for any modular content system.

Newfangled's Christopher Butler wrote the post 'The way you design web content is about to change' (*netm.ag/change-261*). His article inspired me to investigate the Flexible Content Field in WordPress.

Style guide Product style guide for sales force

Modules'. Once you've created a Field Group, you can add fields to it.

Create a single field called 'Flexible Content'. Select `Flexible Content` as the field type. A Flexible Content Field gives you the option to create multiple layouts. Those modules we identified earlier? They map directly to the layouts you will create here. When you create the field it should automatically create your first layout for you.

Go ahead and label the first layout 'Image Text Overlay', with a name 'image_text_overlay'. That name is important because that's how you'll reference the field in your PHP template file. Create a field labeled 'Background Image' (name: 'background_image') and set the `Field Type` to `Image`. Add another field, labeled 'Overlay Text' (name: 'overlay_text') with a `Field Type` of `Text`. That's it for the first layout.

Once you're done with your first layout, hover over the word `Layout` in the left-hand column. Several options should options appear, including `Add New`. Click on that to create your second layout. Label it 'Image and Text Block' (name: 'image_and_text_block'). Add a field labeled 'Featured Image' (name: 'featured_image', field type: `Image`). Add another field, labeled 'Description Text' (name: 'description_text', field type: `Wysiwyg Editor`). This layout maps to the centred image with a text block underneath it.

Each layout maps to a module, so we can craft layout-specific markup in our template

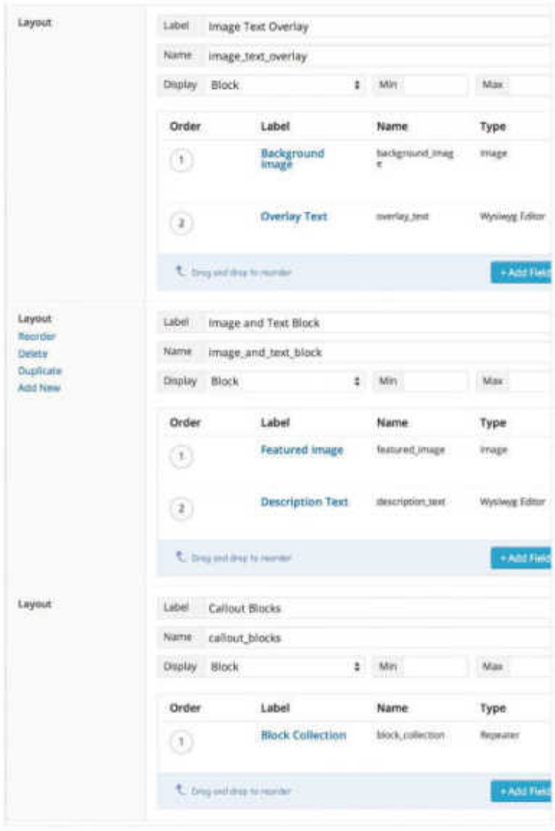
Finally, create your last layout. Label it 'Callout Blocks' (name: 'callout_blocks'). Create a new field labeled 'Block Collection' (name: 'block_collection', field type: `Repeater`). A repeater is a way to create multiple field collections.

In the `Sub Fields` section of the `Repeater` field create a field labeled 'Block Image' (name: 'block_image', field type: `Image`). Create a `Text` field labeled 'Block Text' (name: 'block_text').

FIELDS EXPLAINED

There are our three layouts. If the process of creating field groups and fields is a little confusing, the ACF website has very detailed documentation and videos showing how to create them.

The modules are all pretty similar: they contain an `Image` field, and a `Text` or `Wysiwyg` field for entering



Left A simple detail page wireframe, with three distinct visual modules identified
Right A Flexible Content Field with multiple layouts

text. The only wrinkle is that the last module ontains a `Repeater` field with sub fields, so that we can create multiple field collections within one larger layout.

You may be wondering, why create separate layouts when each layout contains very similar fields? Well, each layout maps to a module in our design system, and this allows us to craft markup in our template that is specific to each layout.

But even more important than the ability to break our markup into separate pieces in our template is the flexibility that layouts provide. Each different layout can be created any number of times. Furthermore, each layout can be dragged in the display order to give your content author the flexibility to decide the best arrangement for the content that makes up each page.

Some pages might only use one or two of the layouts, depending on what content is available for the page. By default ACF will attach your Field Group to the default Post type. If you are creating a big CMS, it's very likely that you'll attach it to a custom post type of your making.

Now let's take a look at it in action. Note the option to create any one of the three layouts by hovering over the '+' symbol. Once your layout has been created, you can also drag your fields to put them them into any desired order.

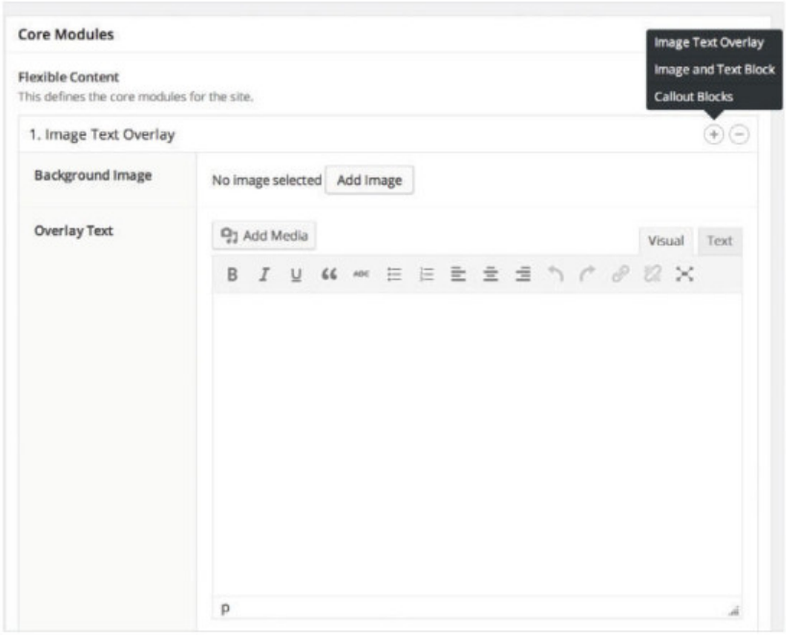
CRAFTING OUR TEMPLATE

So now that you have your content structured into these modular layouts, how do you create a template to display them? Well, it turns out that the PHP template is quite simple. In it you will check for rows in our `Core Module`'s `Flexible Content Field`.

For each row, it can then check the layout type and render the appropriate markup. For rendering the fields themselves we will use the handy ACF methods `the_sub_field()` (*netm.ag/sub-261*) and `get_sub_field()` (*netm.ag/getsub-261*).

I like to generate my themes using underscores.me (*underscores.me*). Your theme should have a `content-single.php` file already present – if not, you can create one. Delete everything in the file and replace it with this code:

```
<?php ?>
<article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
<div class="entry-content">
<?php
if( have_rows('flexible_content') ):
```



Get adaptive Flexible Content Field layouts are easy to create and reorder

```
while ( have_rows('flexible_content') ) : the_row();
if ( get_row_layout() == 'image_text_overlay' ): ?>
<section class="text-overlay">
<h1><?php the_sub_field('overlay_text'); ?></h1>
<?php $background_image = get_sub_field('background_image'); ?>
<img src="<?php echo $background_image['url']; ?>">
</section>
<?php elseif (get_row_layout() == 'image_and_text_block' ): ?>
<section class="text-centered">
<?php $featured_image = get_sub_field('featured_image'); ?>
<img src="<?php echo $featured_image['url']; ?>">
<section class="text-block">
<?php the_sub_field('description_text'); ?>
</section>
</section>
<?php elseif (get_row_layout() == 'callout_blocks' ): ?>
<?php if( have_rows('block_collection') ): ?>
<section class="text-callout">
<?php while( have_rows('block_collection') ): the_row(); ?>
<?php $block_image = get_sub_field('block_image'); ?>
<section class="text-callout-block"><h1><?php the_sub_field('block_text'); ?></h1><img src="<?php echo $block_image['url']; ?>"></section>
<?php endwhile; ?> <!-- block collection while -->
</section>
<?php endif; ?> <!-- block collection if -->
<?php endif; ?>
<?php endwhile; ?>
<?php endif; ?>
</div><!-- .entry-content -->
</article><!-- #post-## -->
```

This template begins by checking if the field `flexible_content` has any rows. If it does, then it loops through each row of the field. Each time through, it checks the `Row Layout` using the `get_row_layout()` method.

You can see that the three names it is checking are the ones you specified for the three layouts: `image_ text_overlay`, `image_and_text_block`, and `callout blocks`. Each of those layouts gets its own (modular) piece of markup. In each markup block you then call `the_sub_ field()` (or `get_sub_field()`, in the case of images that you'll need to set to a variable for printing later).

Now you have a template to render the correct markup for each of your modules, you can style the results with CSS. In many cases I prefer to work out the CSS for each module in a frontend style guide or static HTML mockup, so this step normally involves copying, pasting and testing into the WordPress theme CSS file.

Content authors are given intuitive tools to create layouts and drag them into any order

The CSS I have used is included in the tutorial files. It is fluid (to an extent) but not responsive – you are encouraged to play with it and come up with your own markup/CSS patterns once you understand how the ACF layouts and fields are rendered.

THE FINAL (FLEXIBLE) RESULT

The end result is a page template that supports a flexible design system, on both the authoring side and the rendering side. Content authors are given intuitive tools to create layouts and drag and drop them into the desired order. The result will be well-crafted, consistent markup, ultimately giving the site's users a better experience.

ABOUT THE AUTHOR

PAM SELLE

w: *thewebivore.com*

t: @pamasaur

job: Software engineer

areas of expertise: Building software, generally on the web

q: what's your most impressive hidden talent?
a: My fingers are double-jointed (well, *technically* hypermobile), which freaks people out

HEAD TO HEAD

ANGULAR 2 VS AURELIA

Next-gen JavaScript frameworks Angular 2 and Aurelia are similar in goals and features. **Pam Selle** assesses how they match up

ANGULAR 2	AURELIA
Angular 2 (*angular.io*) is the next version of the popular JavaScript framework AngularJS. It's a complete rewrite of the framework, built looking towards the future of JavaScript and the web, with TypeScript and web components being key parts of its strategy.	Aurelia (*aurelia.io*) is an open-source framework developed as a product of Durandal, Inc. author Rob Eisenberg spent time on the Angular 2 team before leaving to focus on his own framework, which emphasises modern standards and componentisation.
COMPONENTS	
Angular 2 encourages an architecture based around components, or unified blocks of UI markup and behaviour, tracking towards the Web Components spec, and with a view to interoperability with other Web Component libraries like Polymer and X-Tag.	Aurelia also has components, but they're more related to the MVVM pattern, where a component is made up of a view and view-model pair. While the framework is interested in supporting Web Components, its own components are not related to those specifications.
DATA BINDING	
For two-way data binding, you'll need to use the ngModel directive to make an explicit relationship between the view and the data in its component. This is an indicator of the declarative nature of Angular 2, where dependencies are explicitly defined.	Aurelia provides a familiar data-binding style. Defining an attribute in the JavaScript exposes that value to the view, providing two-way data-binding without explicit declarations. Aurelia makes some assumptions about your code, requiring fewer declarations than Angular.
USING THE FUTURE NOW	
Angular 2 is written in TypeScript, intends to be friendly to other compile-to-JS languages, and encourages use of ES2015/2016. Its documentation includes sub-sites for JavaScript, TypeScript and Dart, but documentation varies widely between languages.	Aurelia is written in JavaScript and encourages its users to use ES2015/2016, and also provides a TypeScript version of its starter kit. The documentation has a helpful 'Cheat Sheet' of code snippets to help you get used to the newer syntax.
SUPPORT	
There's some nervousness about Angular 2's long-term viability. Although there is a team working on it within Google, it isn't an official Google product (neither is Angular 1). It's community-supported, and the risks around adopting such a framework remain.	Aurelia is Durandal Inc.'s official product, making it one of the only open-source JavaScript frameworks with commercial backing (Ember.js being the other one). When you need support for Aurelia, you know exactly where to go.

VERDICT

Angular 2 and Aurelia might have a lot in common on their feature lists, but their approaches differ enough to make a difference. If it's a race, Aurelia was first to beta and has stronger documentation. But if you're looking for a framework that includes the world of Web Components and more informed data-binding, go with Angular 2.

FACT FILE

HOW NEW IS AURELIA?
Although Aurelia first entered beta in November 2015, it isn't as new as it may seem: author Rob Eisenberg has said that it's the sequel to the Durandal framework (*durandaljs.com*).

DATA FLOW IN ANGULAR 2
In Angular 2, data flow is managed through Observables, a proposed type in ES2016, and polyfilled using the RxJS 5 library. For Angular users, it represents a shift from largely objects and binding to observing streams of data.

ABOUT THE AUTHOR

RICARDO QUINTAS

w: *ricardoquintas.com*

t: @ricquintas

job: Freelance software engineer

areas of expertise: Full-stack developer

q: what's the first gadget you owned?
a: A pacifier

SITE ANALYSIS

HOW TO DISSECT OTHER PEOPLE'S WEBSITES

Ricardo Quintas explains the best tools and strategies to explore how your fellow developers have constructed their sites

Trying to understand how a website has been built is often a daunting task. After all, you are looking at the end product: the built version. You're dealing with minified and concatenated code that is dynamically loaded depending on the particular agent or device you're are using, which is not an easy thing to grasp. Moreover, you are overloaded with thousands of lines of (sometimes unnecessary) code, some of it created by the developer and some of it generated automatically. This makes life difficult if you want to analyse a particular technical aspect of the site and learn from it.

So is it possible to reverse-engineer a website? And if so, is there any way to make this dissection process easier? Fortunately, the answer to the first question is: yes. As for the second … well, that's where this article comes in.

We will pick up a website and look under the hood to try and understand what is happening and which technologies are being used. The idea here is not to copy the code but to analyse the inner workings of a site so you can learn from it, and hopefully become a better developer.

For the purpose of this article, I have chosen to focus on *foodsense.is*, since it's a comparatively simple site. Once you've mastered the tools and techniques set out here, you can apply them to any type of site, no matter how complex it is.

WIREFRAMES AND DEVICE MODE

It's tempting to jump right into the code, but before doing that, let's try to understand the overall structure of the site and how it generally behaves. I often use Chrome during this process, since it has plenty of useful extensions.

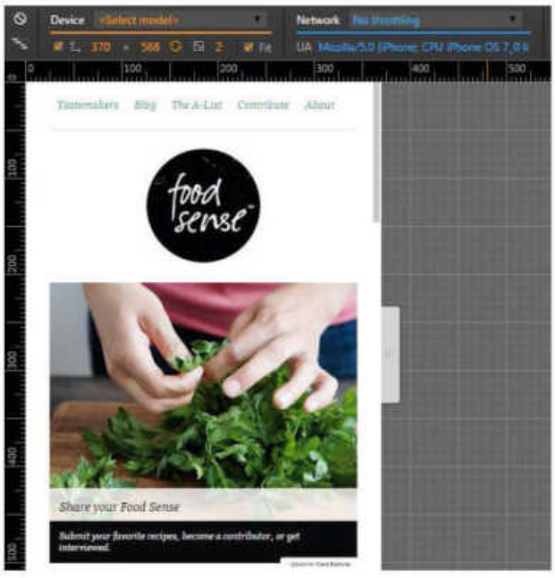
To better understand the structure of a site, I recommend a Chrome extension called Instant Wireframe (*bit.ly/wireframe-264*). This gives you an X-ray view of the site, and is very useful if you want to understand what types of responsive tricks are taking place when you resize your browser window.

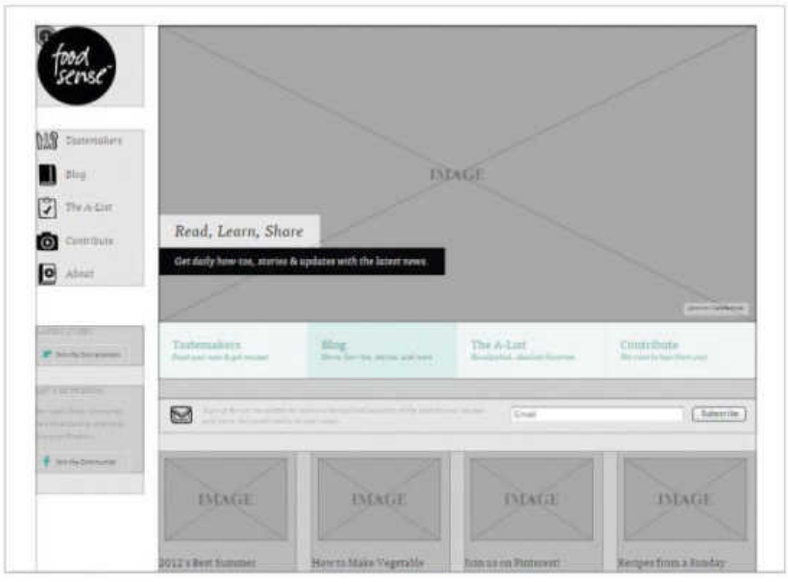
But we can go even deeper than that with Device Mode. Just open DevTools and click on the little 'Toggle device mode' icon. This shows you exactly where the breakpoints are. If you resize *foodsense.is* using this tool, you can confirm that it has breakpoints at approximately 475px, 775px, 975px and 1,175px.

Of course, we're only checking the website through Chrome, running on a desktop. How could we check if it would behave differently if we were running it in a different browser, or on a different device? Here too, Chrome can help you: just enter Device Mode and pick a specific device from the drop-down menu at the top of the screen. After that, all you have to do is refresh the page – try it with websites such as Twitter or LinkedIn and you will see what I mean. By

RESOURCE

Take a look at Ricardo Quintas' cheatsheet for dissecting sites, including some video walkthroughs at *netm.ag/cheatsheet-264*





Left Chrome device mode in action. Try using it with different websites such as Twitter or Linkedin

Right Chrome's Instant Wireframe gives you an X-ray view of the site

using these tools, you can confirm whether you're dealing with a simple responsive site, or whether you have something a little more complex.

Finally, when resizing a website, you can also learn a lot regarding responsive images. For example, if you visit *akqa.com/services* and resize your browser, you will see how the images are swapped, depending on the site's breakpoints.

SHOW ME THE CODE

We have already learned a lot, but we haven't looked at the code yet. If you save a page locally on your machine, you will see that the CSS and JavaScript code is often compressed, making it difficult to understand. You can use online tools such as Unminify (*unminify.com*) to unminify it.

Analysing the inner workings of a site can help you become a better developer

If you use Sublime Text as your code editor, there are some nice plugins to help you as well. For CSS, I use CSS Unminifier (*netm.ag/cssunminifier-264*), and I also use Minify (*netm.ag/minify-264*) to unminify or beautify CSS and JavaScript code.

Please note that plenty of sites have their sources minified into a single file, so even when you unminify the code, it will still exist on a single file. You can try to separate the different parts of the code manually, or you can use the next tool to help you.

ATTACKING STACKS

What if we need to find out which stack was used to build a website? For this, there is a very useful Chrome extension called Wappalyzer (*wappalyzer.com*). It returns a summary of the technologies that were used to build the site. For example, I know *foodsense.is* uses jQuery 1.6.1, Google Analytics, Typekit fonts, Apache web server, the ExpressionEngine (CMS), the CodeIgniter PHP framework – and, of course, PHP itself. Wappalyzer tells us a lot, but not everything. Checking out the Sources tab in Chrome DevTools provides even more information. For example, *foodsense.is* uses the jQuery Cycle plugin for the main image carousel.

Another problem you often face when looking into the source code of a web page is that there is too much of it to understand easily. For example, a lot of websites use frameworks such as Bootstrap, but then use only 10 per cent of the framework's code. To remove the unused code, try UnCSS (*github.com/giakki/uncss*) or the Chrome extension CSS remove and combine (*netm.ag/cssremove-264*). Both can clear unnecessary CSS, making it easier for you to investigate what type of stylesheet tricks are being applied to a specific page.

TRY FOR YOURSELF

Understanding how a site works is both a technique and an art. When you analyse a site, it is always better to focus on a specific element or widget and try to understand how it works, than to try to absorb the entire site at once. Even though the tools covered in this article can help you on that journey, there is no real substitute to diving into the code and getting your hands dirty.

COLOPHON

Future PLC, Quay House, The Ambury, Bath, BA1 1UA +44 (0)1225 442244
@netmag /netmag +netmagazine *flickr.com/photos/netmag* netmag@futurenet.com *net.creativebloq.com*

EDITORIAL

Editor **Oliver Lindberg** *oliver.lindberg@futurenet.com*
Production editor **Ruth Hamilton** *ruth.hamilton@futurenet.com*
Art editor **Mike Brennan** *mike.brennan@futurenet.com*

EDITORIAL CONTRIBUTIONS

Rachel Andrew, Brandon Arnold, Justin Avery, Wes Bos, Espen Brunborg, Ben Callahan, Rosie Campbell, Joe Casabona, Martin Cooper, Gene Crawford, Rob Dodson, Dan Donald, Tom Dougherty, Steve Fisher, Patrick Fulton, Lyza Danger Gardner, Tom Giannattasio, Michael Gooding, Matt Griffin, Patrick Haney, Kevin M Hoffman, Jimmy Jacobson, Una Kravets, Gion Kunz, Jesse Kriss, Mark Llobrera, Sammy Maine, Jenna Marino, Shane Osbourne, Eric Portis, Clarissa Peterson, Ricardo Quintas, Pam Selle, Rebecca Shaw, Julian Shapiro, Noah Stokes, Dan Tello, Vasilis van Gemert, Ray Villalobos, Clark Wimberly, Jennifer Wong, Chloe Wright, Philip Zastrow, Kamil Zieba

ART CONTRIBUTIONS

Bobby Evans, Martina Flor, Jan Kallwejt, Ben Mounsey, Noah Purdy, Joby Sessions, Chandler Williams, Steven Wilson,

MANAGEMENT

Content and marketing director **Nial Ferguson** *nial.ferguson@futurenet.com*
Head of content & marketing, photography, creative and design **Matthew Pierce** *matthew.pierce@futurenet.com,*
Group art director **Rodney Dive** *rodney.dive@futurenet.com*

ADVERTISING Advertising manager **Sasha McGregor** *sasha.mcgregor@futurenet.com*
CIRCULATION Trade marketing manager **Juliette Winyard** *juliette.winyard@futurenet.com*
PRODUCTION Production controller **Nola Cokely** *nola.cokely@futurenet.com*
Production manager **Mark Constance** *mark.constance@futurenet.com*
LICENSING Senior licensing and syndication manager **Matt Ellis** *matt.ellis@futurenet.com*
SUBSCRIPTIONS Subscribe to net online at *myfavouritemagazines.co.uk*

All contents copyright © 2016 Future Publishing Limited or published under licence. All rights reserved. No part of this magazine may be reproduced, stored, transmitted or used in any way without the prior written permission of the publisher. Future Publishing Limited (company number 2008885) is registered in England and Wales. Registered office: Registered office: Quay House, The Ambury, Bath, BA1 1UA. All information contained in this publication is for information only and is, as far as we are aware, correct at the time of going to press. Future cannot accept any responsibility for errors or inaccuracies in such information. You are advised to contact manufacturers and retailers directly with regard to the price and other details of products or services referred to in this publication. Apps and websites mentioned in this publication are not under our control. We are not responsible for their contents or any changes or updates to them.
If you submit unsolicited material to us, you automatically grant Future a licence to publish your submission in whole or in part in all editions of the magazine, including licensed editions worldwide and in any physical or digital format throughout the world. Any material you submit is sent at your risk and, although every care is taken, neither Future nor its employees, agents or subcontractors shall be liable for loss or damage.

Future

Future is an award-winning international media group and leading digital business. We reach more than 49 million international consumers a month and create world-class content and advertising solutions for passionate consumers online, on tablet & smartphone and in print.

Future plc is a public company quoted on the London Stock Exchange (symbol: FUTR).
www.futureplc.com

Chief executive Zillah Byng-Maddick
Non-executive chairman Peter Allen
Chief financial officer Richard Haley

Tel +44 (0)207 042 4000 (London)
Tel +44 (0)1225 442 244 (Bath)

FSC www.fsc.org MIX Paper from responsible sources FSC® C007184

recycle
When you have finished with this magazine please recycle it.

We are committed to only using magazine paper which is derived from well managed, certified forestry and chlorine-free manufacture. Future Publishing and its paper suppliers have been independently certified in accordance with the rules of the FSC (Forest Stewardship Council).

TIPS & TECHNIQUES

INSPIRATION

TOOLS

HANDS-ON TUTORIALS

THE RESPONSIVE WEB DESIGN HANDBOOK

VOLUME II

In this special edition from the makers of net magazine, you'll find everything you need to know about the latest responsive design and development techniques, from cutting-edge layout options and responsive images to emails and WordPress portfolios

Master responsive design today!